UQAC
Université du Québec
à Chicoutimi

MÉMOIRE

PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

AMADOU MAKHTAR TALL

SUR L'ARITHMÉTIQUE DES POLYOMINOS

JANVIER 2015

REMERCIEMENTS

En préambule à ce mémoire, je voudrais sincèrement remercier M. Alexandre Blondin Massé pour avoir bien voulu diriger mes travaux de recherche. Comme son étudiant, j'ai beaucoup apprécié son sens de la rigueur, de l'écoute et du partage. Il m'a soutenu et appuyé dans la réalisation de ce mémoire. Il m'a fait découvrir le monde fascinant de la recherche et m'a introduit à la combinatoire des mots et aux mathématiques-informatique.

Je voudrais également exprimer mes sincères remerciements à mes parents qui ont toujours été présents pour moi.

J'exprime ma reconnaissance ultime à notre Seigneur Tout Puissant, pour m'avoir permis d'arriver à cette étape de ma vie.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

RÉSUMÉ

Ce mémoire de maîtrise se consacre à l'étude des figures discrètes, un sujet à l'intersection de la combinatoire des mots et la géométrie digitale. Décrite simplement, une *figure discrète* est un assemblage fini de pixels joints côté par côté. Toute figure discrète se décompose en composantes connexes appelées *polyominos*, un objet très connu en combinatoire et en théorie des jeux.

Dans sa thèse, en 2008, Provençal propose quelques problèmes ouverts sur les polyominos et introduit ainsi le concept de polyominos *premiers* et *composés*. Jusqu'à maintenant, on en sait très peu sur cette notion de primalité. En effet, à notre connaissance, le seul document qui y fait référence est un article publié en 2012 par Blondin Massé, Garon et Labbé dans lequel les auteurs résolvent une conjecture de Provençal en s'appuyant sur la notion de polyominos premiers et composés.

Dans ce mémoire, nous explorons plus en détails ce sujet. En plus de fournir une définition et un cadre plus formel, nous proposons un algorithme polynomial (par rapport au périmètre du polyomino) permettant de décomposer celui-ci en un produit de polyominos premiers. Nous discutons également des implications potentielles de ces idées en cryptographie.

CHAPITRE 1

INTRODUCTION

Ce mémoire s'inscrit dans le domaine de la géométrie discrète, c'est-à-dire la modélisation d'objets continus à l'aide d'une représentation discrète. Cette branche de l'informatique a connu un essor remarquable avec l'avènement des ordinateurs : étant donné la grande quantité de données numériques de nature géométrique, il est fondamental de développer des algorithmes traitant efficacement ces données et les problèmes qu'elles soulèvent. À titre d'exemple, la géométrie discrète est utilisée en synthèse d'images [Glassner, 1995], dans le contrôle à distance, en télédétection par satellite [Dagorne, 1989], en morphologie [Lu et al., 2003], en météorologie [Richardson, 2007] et en géomorphologie [Dikau, 1989].

En géométrie discrète, on propose habituellement des représentations discrètes qui s'appliquent à un traitement numérique tout en conservant les propriétés essentielles de leurs antécédents continus. Dans cette perspective, Freeman a proposé en 1961 une façon pratique d'encoder tout chemin dans le plan discret comme une suite des quatre déplacements élémentaires $\{\rightarrow, \uparrow, \leftarrow, \downarrow\}$ par l'alphabet $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$. Ce chemin peut alors être représenté par un mot, ce qui permet d'étudier l'objet géométrique qu'il encode d'un point de vue combinatoire. Une *figure discrète* peut donc être vue comme un ensemble de mots munis chacun d'un point de départ, où le mot en question décrit le contour d'une région fermée.

Il convient de mentionner que, suite au développement de ces nouveaux champs de recherche sur le traitement d'images numérisées, la communauté scientifique s'est organisée en associations, parmi lesquelles on compte l'association scientifique à but non lucratif, créée en janvier 1978, appelée **International Association for Pattern Recognition** dont le mandat est de promouvoir la reconnaissance par motifs [IARP, 2014]. La reconnaissance par motifs est utilisée dans plusieurs contextes, tels que la robotique. Il s'agit donc d'un domaine qui regroupe une importante communauté de chercheurs, dont le sujet a été l'objet de plusieurs conférences. Par exemple, la conférence internationale DGCI (**Discrete Geometry for Computer Imagery**) [DGCI, 2014] est centrée sur les aspects théoriques de la reconnaissance de motifs. PRIB (**Pattern Recognition in Bioinformatics**) [PRIB, 2014] traite de la présentation, de la discussion et de l'application de méthodes de reconnaissance par motifs en bio-informatique.

Le nom *polyomino* a été proposé par Golomb en 1953 [Golomb, 1996], puis ultérieurement popularisé par Gardner, très actif dans les mathématiques récréatives durant une bonne partie du 20e siècle. Les polyominos sont très connus du grand public. En effet, il suffit de penser au jeu Tétris qui consiste à remplir une matrice par des tétraminos (voir figure 1.1).

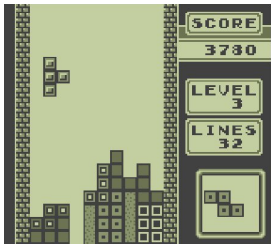


Figure 1.1: Le jeu vidéo Tetris, qui a connu une popularité immédiate dans le monde entier à la fin des années 80 <http://www.le-serpent-retrogamer.org/un-an-une-tof-1984-tetris/>

Il existe en outre des jeux de société très populaires utilisant le polyomino comme unité de base, tels que « Caminos », qui est un jeu d'orientation dans l'espace consistant à tracer

un passage, d'un bord à l'autre du plateau de jeu, avec des blocs de même couleur (voir figure 1.2). De la même façon, le jeu Kaléidoscope, un jeu de structuration spatiale (voir figure 1.3) consiste à reproduire un motif (un rectangle par exemple) à partir de 18 pièces, chacune unique.

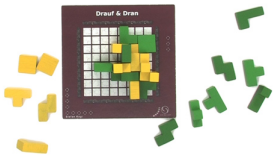


Figure 1.2: Le jeu « Caminos ». La règle est simple, être le premier à relier de manière ininterrompue deux côtés opposés du plateau avec des pièces de la couleur choisie [CAMINOS, 2014].

Figure 1.3: Un kaléidoscope (kalos signifie « beau », eidos « image », et skopein « regarder »). À la base c'est un casse-tête pour jouer en solitaire. Le but du jeu est de reconstituer le plateau selon un plan déterminé avec des pièces qui sont des polyominos [KALEIDOSCOPE, 2014].

Le célèbre algorithme « Dancing links » (liens dansants) proposé par D. Knuth en 2000 a été initialement conçu dans l'objectif de résoudre un problème de pavage d'une forme à l'aide de polyominos ou de *polycubes* (la version tridimensionnelle des polyominos). Il consiste à résoudre l'assemblage d'un puzzle par pavage efficace de polyominos [Knuth, 2000] (l'implémentation d'un tel solveur peut être trouvée dans « Tiling solver in Sage, 2011 » [Labbé, 2011]).

1.1 REVUE DE LITTÉRATURE

Plusieurs chercheurs se sont intéressés à la modélisation des figures discrètes. Comme il s'agit d'un domaine vaste, nous nous concentrons sur les plus pertinents d'entres elles pour ce mémoire.

Un des exemples de la littérature qui établit un lien entre combinatoire des mots et géométrie discrète est dû à Vuillon et Berthé qui ont proposé une correspondance entre les plans discrets et des mots bidimensionnels sur un alphabet à trois lettres [Berthé et Vuillon, 2000]. En dimension un, ce sont sans aucun doute les *mots sturmiens* qui en sont les meilleurs exemplaires, puisqu'ils décrivent une *discrétisation* d'une droite. Une propriété importante de ces mots est qu'ils présentent de nombreux facteurs palindromiques, c'est-à-dire des mots qui se lisent de la même façon dans les deux directions (comme *radar*). Or, les *palindromes* sont des indicateurs précieux de la structure de plusieurs mots et permettent des interprétations notamment en géométrie discrète. En effet, les mots sturmiens, étant riches en palindromes, induisent des symétries qui se traduisent sur les objets géométriques qu'ils représentent.

En 2005, Brass et al. considèrent un grand choix de problèmes rencontrés par les chercheurs dans le domaine de la géométrie discrète et leurs résolutions par l'utilisation du modèle combinatoire [Brass et al., 2005]. Leur ouvrage « Research problems in discrete geometry » est le résultat d'un projet de 25 ans, initié par Leo Moser. On y décrit les frontières et l'avenir de la recherche sur la géométrie discrète. Il s'agit d'une collection de plus de 500 problèmes ouverts dans ce domaine. Les différents chapitres offrent un large aperçu sur le domaine, avec des détails historiques et les solutions partielles les plus reconnues apportées à ces problèmes. Certains problèmes sont notoirement difficiles et intimement liés à des questions profondes dans d'autres domaines des mathématiques.

Le *mot de contour* peut être vu comme un codage des polyominos par un alphabet à quatre lettres décrivant les quatre déplacements élémentaires : droite, haut, gauche et bas. Les lettres $\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}$ encodent respectivement ces déplacements et $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$ est appelé *alphabet de Freeman*. L'avantage principal de cette représentation est qu'il est alors possible de considérer l'alphabet comme le groupe $\mathbb{Z}_4$. De plus, on peut extraire plusieurs informations sur le polyomino simplement en considérant son mot de contour. À titre d'exemple, Brlek et al. ont proposé en 2009 un algorithme linéaire (en temps et en espace) pour détecter les points d'intersections de chemins dans un espace discret dimension de $\mathbb{Z}^d$ [Brlek et al., 2009a]. La puissance de l'algorithme repose principalement sur l'utilisation d'un arbre radix. En outre, cette structure de données peut sans aucun doute s'adapter à d'autres algorithmes, tels que l'intersection et la réunion de figures discrètes [Morrison, 1968].

Les polyominos sont aussi des objets de base dans l'étude des pavages. Des pavages de formes très variées sont utilisés depuis l'Antiquité pour l'aménagement et la décoration (carrelages, frises, peintures). Aussi, on rencontre dans la nature des exemples remarquables de pavages réguliers et irréguliers, comme dans les nids d'abeille ou dans les réseaux cristallins. La recherche permet de comprendre ces matériaux afin de les utiliser dans la fabrication de matières synthétiques, d'alliages ou de matériaux ultra-légers.

Intuitivement, une *tuile* est une région plane dont la frontière est un polygone fermé. Nous savons, depuis 1970, que le problème du pavage d'un plan par des polyominos libres (polyominos qui peuvent subir une rotation ou une réflexion) choisis à partir d'un ensemble fini est indécidable [Golomb, 1970]. Et même quand on se restreint à une seule tuile, la décidabilité du problème du pavage est inconnue. De même, le problème de savoir si un polyomino donné pave un autre polyomino est NP-complet [Moore et Robson, 2001]. Dans le cas où aucune réflexion ou rotation d'un polyomino libre n'est permise, le problème devient considérablement plus simple. Non seulement cela est décidable, mais on peut le déterminer en temps

polynomial : une borne $O(n^2)$ est démontrée dans [Gambini et al., 2007], réduite à $O(n)$ dans [Brlek et Provençal, 2006].

En effet, l'idée de base vient d'un article de Beauquier et Nivat qui ont caractérisé ces objets par la forme de leur contour. Ils démontrent entre autres que ces polyominos peuvent nécessairement être divisés en quatre ou six morceaux parallèles par paires [Beauquier et Nivat, 1991]. En 2006, Brlek et Provençal considèrent le problème de déterminer si un mot de contour d'une figure discrète donné pave le plan par translation [Brlek et Provençal, 2006].

D'un point de vue algorithmique, soit n la longueur du mot de contour d'un polyomino p, la caractérisation Beauquier-Nivat fournit un algorithme naïf déterminant si p est pas défini en $O(n^4)$. Gambini et Vuillon utilisent cette caractérisation et proposent en 2007 un algorithme pour décider si un polyomino pave le plan donné par translations en $O(n^2)$ [Gambini et al., 2007]. Par la suite, en 2009, pour les polyominos parallélogrammes, Brlek et al. proposent un algorithme de détection en temps linéaire [Brlek et al., 2009b], améliorant l'algorithme quadratique de Gambini et Vuillon. Leur approche est inspirée par l'algorithme linéaire de Gusfield et Stoye pour détecter les répétitions en tandem dans un mot [Gusfield et Stoye, 2004] et par l'algorithme linéaire utilisé pour détecter des répétitions avec gaps tel que montré dans Lothaire [Lothaire, 1983]. Plus précisément, par une utilisation intelligente de l'arbre des suffixes [Gusfield, 1997], il performe en temps constant un prétraitement linéaire sur les mots des polyominos parallélogrammes u et v. Dans le cas des polyominos hexagonaux, la complexité est linéaire si le nombre de répétitions de sous-motifs est borné par une constante [Brlek et al., 2009b].

Dans [Blondin Massé et al., 2013], les auteurs examinent le problème de l'énumération des polyominos appelés *doubles parallélogrammes* (voir figure 1.4), qui sont des polyominos engendrant deux pavages parallélogrammes distincts. Afin de prouver l'un des principaux

résultats de l'article, les auteurs se basent sur le concept de polyominos premiers et composés, introduit par Provençal dans sa thèse de doctorat [Provençal, 2008].

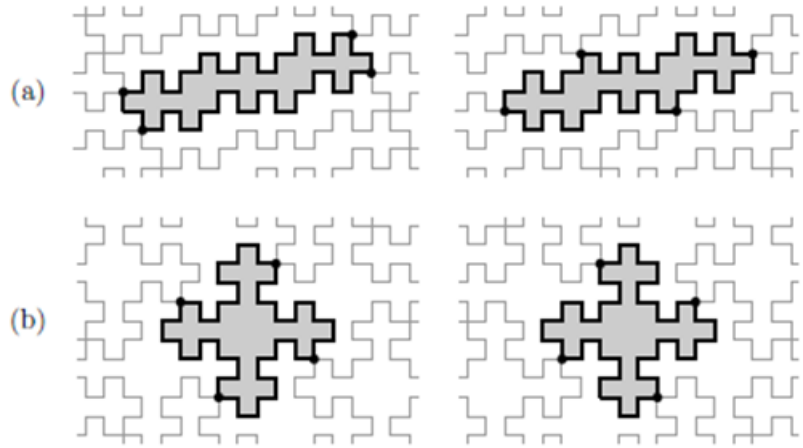


Figure 1.4: Deux doubles parallélogrammes et le pavage qu'ils induisent. Les points noirs indiquent les points partagés par quatre copies de la tuile. (a) Une tuile de Christoffel. (b) Une tuile de Fibonacci (voir [Blondin Massé et al., 2009] pour plus de détails).

1.2 PROBLÉMATIQUE

On dit d'un polyomino qu'il est *premier* s'il est impossible de le paver à l'aide d'un autre polyomino différent du carré unité. Autrement, il est dit *composé*. De façon équivalente, cela veut dire qu'il ne peut être obtenu à partir d'une autre figure plus petite en remplaçant les pas élémentaires par des chemins plus complexes.

En combinatoire des mots, cela signifie qu'on ne peut obtenir le mot de contour à partir d'un autre en lui appliquant un *morphisme homologue*, soit une substitution telle que les chemins codant la figure discrète sont les mêmes à ordre de parcours inversé.

Afin d'étudier les propriétés d'un polyomino premier, il faut d'abord concevoir un algorithme permettant de décider s'il l'est. Il s'agit d'une des contributions principale de ce mémoire.

En outre, nous proposons un algorithme qui décompose n'importe quel polyomino en polyominos premiers. Nous proposons également une généralisation du théorème fondamental de l'arithmétique pour les polyominos.

Le premier chapitre est consacré à la définition des objets combinatoires et géométriques qui seront étudiés et employés par la suite. Les définitions et la notation relatives à la combinatoire des mots sont en majeure partie tirées des livres de M. Lothaire.

Le chapitre 2 est consacré à l'étude d'une classe de morphismes particuliers appelés *morphismes homologues*. Celle-ci permet alors de définir la notion de polyominos premiers et composés. On y étudie aussi les aspects algorithmiques soulevés par ces problèmes.

Le chapitre 3 est consacré à l'évaluation empirique des concepts théoriques de nos deux algorithmes de factorisation proposés dans le chapitre 2.

Les résultats de ce mémoire ont été présentés en mars 2014 à la conférence **Language and Automata Theory and Applications (LATA)**, une conférence annuelle internationale organisée par le Groupe de recherche sur la linguistique mathématique (GRLMC) [Blondin Massé et al., 2014] et dont les actes sont publiés par Springer.

CHAPITRE 2

DÉFINITIONS ET NOTATIONS

Dans ce chapitre, on introduit la notation et des définitions relatives à la combinatoire des mots et aux polyominos. Le lecteur intéressé à en savoir davantage est référé à M. Lothaire [Lothaire, 1983].

2.1 MOTS

Un *alphabet A* est un ensemble fini dont les éléments sont appelés *lettres* (ou *symboles*). Nous appelons *mot* sur un alphabet A toute suite finie de lettres. Plus formellement, un *mot fini* w sur un alphabet A est une suite finie $(w_1, w_2, ..., w_n)$ d'éléments de A, où $n \in \mathbb{N}$. Afin d'alléger la notation, on écrit $w = w_1 w_2 \cdots w_n$. Pour $i = 1, 2, \ldots, n$, on note par $w[i]$ la lettre du mot w à l'indice i (souvent, pour des raisons arithmétiques, on commence l'indexation par 0). L'entier n est appelé la *longueur* de w, notée $|w|$. Il existe un unique mot w tel que $|w| = 0$. Ce mot est appelé *mot vide* et est noté ε.

Exemple 1. *Soit* $A = \{a, y, b, x\}$ *un alphabet. Alors* ε*, x, a et xaxa sont des mots sur l'alphabet A.*

On désigne par A^n l'ensemble des mots de longueur n sur A, où $n \in \mathbb{N}$. L'ensemble des mots

de longueur quelconque sur A, noté par A^* est défini par :

$$A^* = \bigcup_{n\geq 0} A^n.$$

Soient deux mots $u = u_1u_2\cdots u_m$ et $v = v_1v_2\cdots v_m$ sur A, avec $m,n \in \mathbb{N}$. On appelle *concaténation* de u et v le mot $u\cdot v = u_1u_2\cdots u_mv_1v_2\cdots v_n$. Souvent, on écrit simplement uv plutôt que $u\cdot v$. La concaténation de deux mots est une opération associative sur A^*, de sorte que $(A^*,\cdot)$ est un monoïde, appelé *monoïde libre*, dont l'élément neutre est ε.

Soit w et y deux mots sur l'alphabet A. On dit que y est un *facteur* de w s'il existe deux mots u et v tels que $w = uyv$. En particulier, si $u = \varepsilon$, alors y est appelé *préfixe* de w et si $v = \varepsilon$, alors on dit que y est un *suffixe* de w. Un facteur, un préfixe ou un suffixe y de w est dit *propre* si $x \neq \varepsilon, y$. Ces notions induisent naturellement des relations sur les mots. On écrit $x \preceq_{fact} y$, $x \prec_{fact} y$, $x \preceq_{pref} y$, $x \prec_{pref} y$, $x \preceq_{suff} y$ et $x \prec_{suff} y$ si x est respectivement un facteur, un facteur propre, un préfixe, un préfixe propre, un suffixe ou un suffixe propre de y. Les relations $\preceq_{fact}$, $\preceq_{pref}$, $\preceq_{suff}$ sont des relations d'ordre partiel. On note respectivement $\mathrm{Fact}(w)$, $\mathrm{Pref}(w)$ et $\mathrm{Suff}(w)$ d'un mot w donné l'ensemble des facteurs, préfixes et suffixes.

Exemple 2. *Soit $w = algorithmique$. Le mot algo est un préfixe de w, que est un suffixe de w et rithmi est un facteur de w.*

Soit w un mot indicé entre 1 et n. On dit que le nombre i est une *occurrence* de u si $w = xuy$ pour certains mots x et y, avec $|x| = i-1$. On désigne par $|w|_u$ le nombre d'occurrences de u dans w. En outre, u est dit *unioccurrent* dans w si $|w|_u = 1$.

La *n-ième puissance* d'un mot non vide w, notée w^n, est donnée par $w^n = ww\cdots w$ (n fois). On dit que w est *primitif* s'il n'existe aucun mot u tel que $w = u^n$, pour un certain entier n. En particulier, le *carré* de w est donné par w^2.

Exemple 3. $(aba)^3 = abaabaaba$ *et* $aba^3 = abaaa$.

Exemple 4. *Le mot abaab est primitif. Les mots* ε *et* $ababab = (ab)^3$ *ne sont pas primitifs.*

Deux mots u et v de A^* sont *conjugués*, et on écrit $u \equiv v$ s'il existe deux mots x et y tels que $u = xy$ et $v = yx$. La relation de conjugaison $\equiv$ est une relation d'équivalence et la classe du mot w est dénotée par $[w]$. On dit alors que $[w]$ est un *mot circulaire*.

Exemple 5. *Les mots* $u = ababa$ *et* $v = abaab$ *sont conjugués. En effet, il suffit de prendre* $x = ab$ *et* $y = aba$ *et on voit que* $u = ababa = xy$ *et* $v = abaab = yx$.

L'*image miroir* d'un mot $w \in A^*$ est le mot $\widetilde{w}$ défini par $\widetilde{w} = w[n]w[n-1]...w[1]$. Autrement dit, l'ordre des lettres est renversé. On dénote par $\textsc{Fst}(w)$ et $\textsc{Lst}(w)$ respectivement la première et la dernière lettre du mot w. L'ensemble des lettres qui apparaissent dans un mot w est noté $\mathrm{Alph}(w)$. Par exemple, $\mathrm{Alph}(maammamammamam) = \{m, a\}$.

Soit $\varphi : A^* \to B^*$ une fonction, où A et B sont deux alphabets. On dit que φ est un *morphisme* s'il préserve la concaténation, c'est-à-dire que $\varphi(uv) = \varphi(u)\varphi(v)$, pour n'importe quels $u, v \in A^*$. D'autre part, on dit que φ est un *antimorphisme* si $\varphi(uv) = \varphi(v)\varphi(u)$ pour n'importe quels $u, v \in A^*$. L'application image miroir $\widetilde{\cdot}$ est un antimorphisme.

Soit la fonction $f : A \to A$, une involution est une application bijective telle que $f \circ f = id_A$. Supposons que $A = \{a, b\}$. Le *complément* d'un mot $w \in A^*$, dénoté par $\overline{w}$, est le mot obtenu par l'application du morphisme échangeant les lettres de w, c'est-à-dire que $\overline{\cdot}$ est le morphisme défini par $\overline{a} = b$ et $\overline{b} = a$. Il est clair que l'opération de complémentation est une involution.

La complexité (*factorielle*) d'un mot est la fonction indiquant, pour chaque naturel n, le nombre de mots de longueur n, c'est-à-dire la fonction

$$C_w(n) : \mathbb{N} \to \mathbb{N} : n \mapsto |Fact_n(w)|.$$

2.2 POLYOMINOS

Les polyominos sont des assemblages de n carrés (pixels) finis congrus joints côté par côté sans trou. C'est une généralisation naturelle des dominos (voir figure 2.1).

Figure 2.1: Un pavage partiel utilisant des polyominos de 4 cellules, aussi appelés tétraminos.

La *grille carrée* est l'ensemble $\mathbb{Z}^2$. Une cellule est une unité carré dans $\mathbb{R}^2$ dont les coins sont des points de $\mathbb{Z}^2$. Nous dénotons par $c(i,j)$ la cellule $c(i,j) = \{(x,y) \in \mathbb{R}^2 |\ i \leq x \leq i+1, j \leq y \leq j+1\}$. L'ensemble de toutes les cellules est dénoté par C.

Définition 1. *Deux cellules c et d* $\in C$ *sont dites* 4-adjacentes *si elles ont exactement une côté en commun, c'est-à-dire si elles se touchent par une arête (voir figure 2.2).*

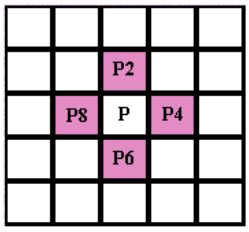


Figure 2.2: Les quatre cellules 4-adjacentes (P_2, P_4, P_6, P_8) de la cellule P.

Clairement chaque cellule $c(i,j)$ a exactement quatre cellules voisines $c(i+1,j)$, $c(i,j+1)$, $c(i-1,j)$ et $c(i,j-1)$.

Définition 2. *On appelle* chemin 4-connexe *(voir figure 2.3) entre deux cellules c et d toute suite de cellules* $(c_1, c_2, \ldots, c_n)$ *telle que* $c = c_1$, $d = c_n$ *et* c_i *et* C_{i+1} *sont* 4*-adjacentes pour* $i = 1, 2, \ldots, n-1$. *Un ensemble de cellules E est dit* 4*-connexe si, pour toute paire de cellules* $c, d \in E$, *il existe un chemin* 4*-connexe entre c et d n'utilisant que des cellules de E. Un ensemble connexe est parfois appelé* région 4-connexe *ou simplement* région.

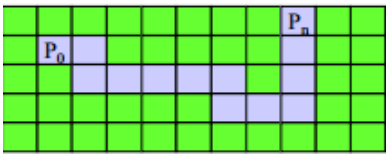


Figure 2.3: Chemin 4-connexe

La figure 2.4 illustre des ensembles 4-connexes et la figure 2.5 illustre des exemples d'ensembles non connexes.

Figure 2.4: Deux ensembles connexes.

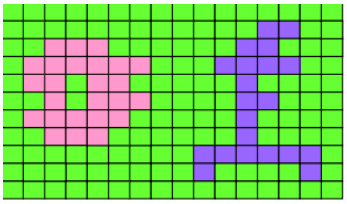
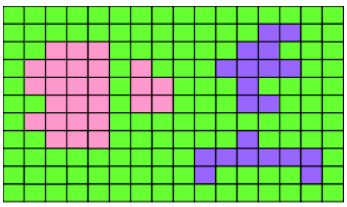


Figure 2.5: Deux ensembles non connexes (les cellules de même couleur ne sont pas toutes 4-connexes.



Les isométries usuelles (translation (θ), rotation (ρ) et réflexion (σ)) sont naturellement définies pour les régions 4-connexe. Dans certains cas, il est commode de considérer deux régions équivalentes à isométrie près. On écrit $R \sim_\theta S$ si S est une copie translatée de R. Il s'agit clairement d'une relation d'équivalence. Dans le même esprit, on écrit $R \sim_\rho S$ si S peut être obtenu de R par une combinaison de translations et de rotations. Finalement, on écrit $R \sim_\sigma S$ si S est obtenu de R par une combinaison de translations, de rotations et de réflexions. Les classes d'équivalence des relations $\equiv_\theta$, $\equiv_\rho$ et $\equiv_\sigma$ sont appelées respectivement *polyomino fixe*, *polyomino unilatéral* et *polynomino libre*. Les polyominos libres sont représentés à la figure 2.6.

Une autre notion importante est celle de trou. Soit R une région. On définit le *complément* de R par $\overline{R} = C - R$. Un *trou* de R est une région $T \subseteq \overline{R}$ qui est 4-connexe, maximale et finie. Si R est une région finie, alors elle peut avoir un nombre fini de trous et exactement une région de $\overline{R}$ est infinie.

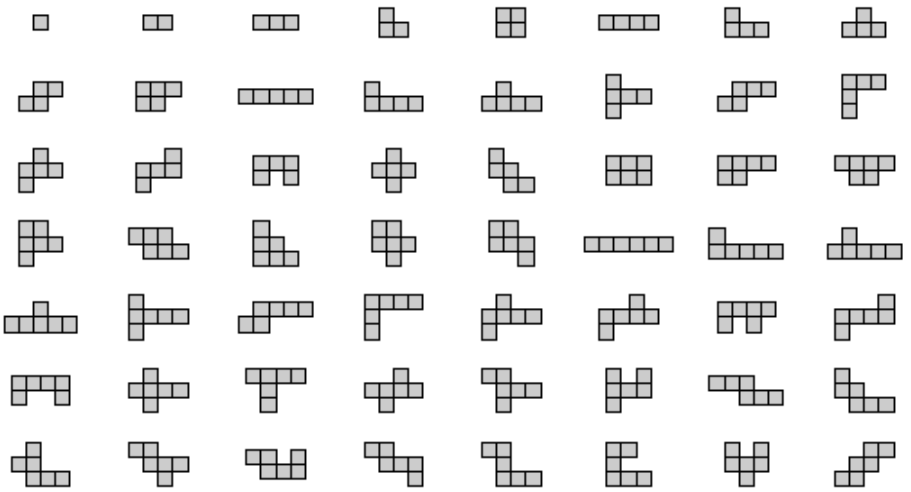


Figure 2.6: Tous les polyominos libres de c cellules, pour $c \in \{1,2,3,4,5,6\}$

2.3 MOT DE CONTOUR

Un alphabet fondamental en géométrie digitale est le code de Freeman $\mathscr{F} = \{\mathbf{0},\mathbf{1},\mathbf{2},\mathbf{3}\}$. Pour des raisons pratiques, il est convenable de le considérer comme un groupe additif modulo 4. Il permet de coder un chemin le long du contour dans un repère absolu à partir d'une origine fixée. Il peut y avoir 4 ou 8 directions permises selon la situation (voir figure 2.7).

Le codage d'un contour se fait donc de la façon suivante :

1. On indique les coordonnées absolues du point de départ.
2. On donne la liste des codes de déplacement d'un point du contour au suivant dans l'espace discret.

La figure 2.8 nous donne un exemple d'approximation d'une courbe par le code de Freeman.

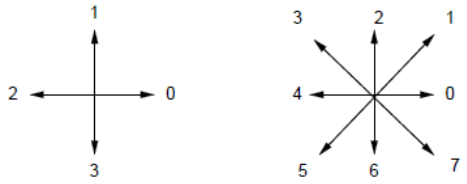


Figure 2.7: Code de Freeman en 4**-connexe (à gauche) et en** 8**-connexe (à droite)** <http://perso.telecom-paristech.fr/~tupin/TDI/poly_formes.pdf>

Figure 2.8: Code de Freeman. Approximation de la courbe A **par la chaîne** x **(en** 4**-connexe) et par la chaîne** y **(en** 8**-connexe)** <http://perso.telecom-paristech.fr/~tupin/TDI/poly_formes.pdf>

Dans ce mémoire, nous nous concentrons sur l'alphabet de Freeman $\mathscr{F}$ de 4 lettres. Un chemin (voir figure 2.9) p est un mot sur $\mathscr{F}$, c'est-à-dire un élément de $\mathscr{F}$.

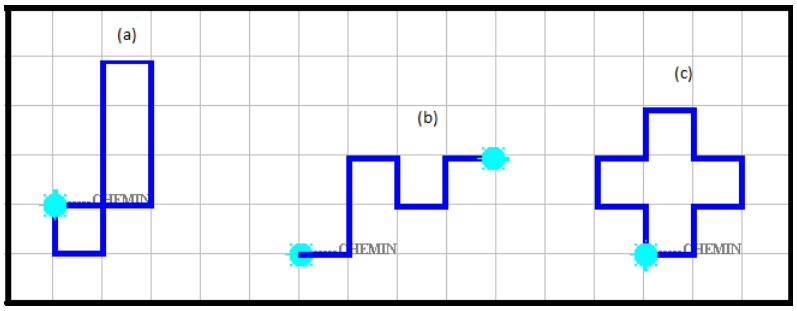


Figure 2.9: Exemples de chemins. (a) Un chemin fermé qui n'est pas auto-évitant, (b) un chemin auto-évitant qui n'est pas fermé et (c) un mot de contour qui est fermé et auto-évitant.

Un chemin p est un mot sur $\mathscr{F}$, c'est-à-dire un élement de $\mathscr{F}^*$. On dit de p qu'il est *fermé* si $|p|_0 = |p|_2$ et $|p|_1 = |p|_3$. Un sous-chemin q de p est simplement un facteur de p. Le chemin p est dit *auto-évitant* s'il ne possède aucun sous-chemin propre fermé. Un *mot de contour* est un chemin auto-évitant et fermé.

Dans de nombreuses situations, il est commode de représenter un polyomino sans trou par son mot de contour, qui à son tour, est facilement représenté par un mot sur l'alphabet de Freeman. La représentation des polyominos par mot de contour offre les caractéristiques intéressantes suivantes :

1. Elle est simple, unique à conjugaison et orientation près.
2. Dans la plupart des cas, l'aire d'un polyomino est quadratique par rapport à son périmètre. Conséquemment cela fait que la représentation du contour, de longueur égale à la longueur du périmètre, occupe ainsi un espace plus compact.
3. Les principes et théories issus de la combinatoire des mots peuvent y être appliqués.
4. De nombreuses propriétés de symétrie, d'isométrie et de convexité d'un polyomino donné sont détectées facilement par l'étude et l'analyse simple du contour. Par exemple,

si le contour est fermé et auto-évitant, on le définit comme un mot de contour.

Les isométries de base de la grille discrète (rotations d'angle multiple de $\frac{\pi}{2}$ et réflexions) se définissent naturellement à l'aide de morphismes appliqués sur les lettres.

Rotations. Le morphisme $\rho : \mathscr{F}^* \to \mathscr{F}^*$ défini par

$$\rho : x \mapsto (x+1) \bmod 4,$$

agit sur le chemin comme une rotation d'angle $\frac{\pi}{2}$. Par conséquent, le morphisme ρ^i est une rotation d'angle $\frac{i\pi}{2}$.

La rotation d'angle π (voir figure 2.10) ρ^2 est aussi notée $\bar{\cdot}$, c'est-à-dire la rotation définie par $\rho^2(0) = \overline{0} = 2, \rho^2(1) = \overline{1} = 3, \rho^2(2) = \overline{2} = 0, \rho^2(3) = \overline{3} = 1$.

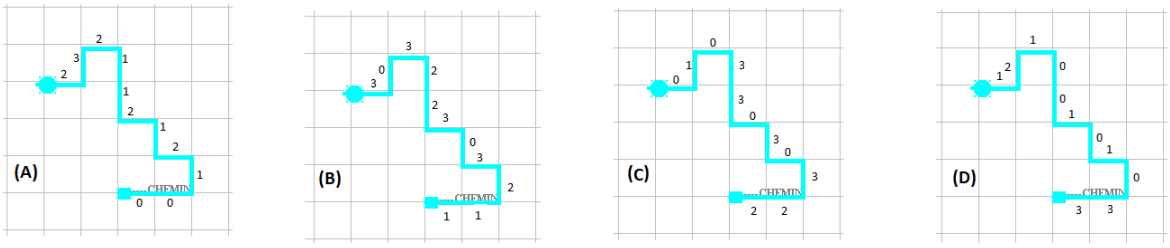


Figure 2.10: Le chemin $w = 00121211232$ et ses rotations $\rho(w)$, $\rho^2(w)$ et $\rho^3(w)$.

Symétries. Étant donné $i \in \{0,1,2,3\}$, soit le morphisme $\sigma_i : \mathscr{F}^* \to \mathscr{F}^*$ défini par

$$\sigma_i : x \to (i-x) \bmod 4.$$

Donc $\sigma_0, \sigma_1, \sigma_2, \sigma_3$ agissent respectivement sur les chemins par symétrie (voir figure 2.11) par rapport aux axes $y = 0$, $x = y$, $x = 0$ et $x = -y$.

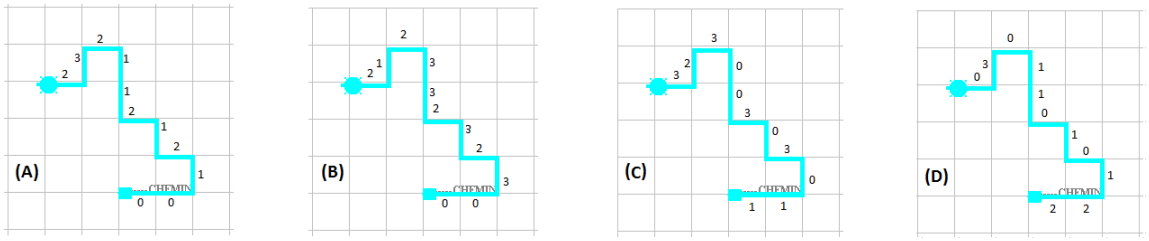


Figure 2.11: Le chemin $w = 00121211232$ et les symétries $\sigma_0(w)$, $\sigma_1(w)$ et $\sigma_2(w)$.

Parcours inversé. Rappelons qu'un antimorphisme sur un alphabet A est une application provenant de la fonction miroir $\varphi : A^* \to A^*$ vérifiant $\varphi(uv) = \varphi(v)\varphi(u)$ pour n'importe quels $u, v \in A^*$. On dit d'un antimorphisme qu'il est involutif si $\varphi^2 = \mathrm{Id}$. Il est facile de vérifier que tout antimorphisme involutif φ peut-être décomposé en $\varphi = \sigma \circ R = R \circ \sigma$ où σ est une involution sur l'alphabet A. L'antimorphisme $\widehat{\cdot} : \mathscr{F}^* \to \mathscr{F}^*$ est défini par

$$\widehat{\cdot} = \rho^2 \circ \tilde{\cdot} = \tilde{\cdot} \circ \rho^2.$$

D'un point de vue géométrique, l'antimorphisme $\widehat{\cdot}$ correspond à inverser la direction de parcours d'un chemin (voir figure 2.12). On dit alors que w et $\widehat{w}$ sont des chemins *homologues*.

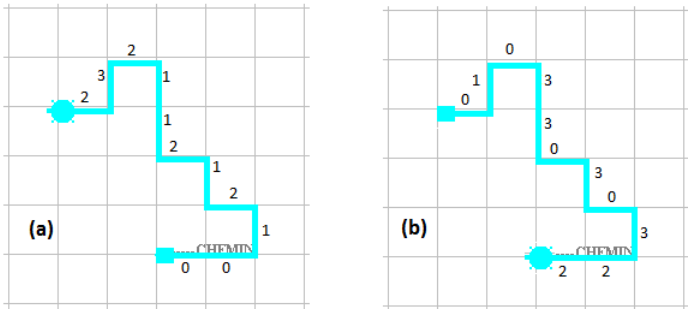


Figure 2.12: Les chemins homologues (a) $w = 00121211232$ et (b) $\widehat{w} = 01033030322$

2.4 VIRAGES ET ENROULEMENTS

Comme mentionné auparavant, le code de Freeman décrit des chemins selon les quatre directions élémentaires *haut*, *bas*, *gauche*, *droit*. Cependant, il existe une autre représentation pratique des chemins discrets basée sur la notion de virages. À cette fin, nous introduisons l'opérateur Δ sur $\mathscr{F}$ de la façon suivante. Soit $w = w_1 w_2 ... w_n$ un chemin de longueur $n \geq 2$. Le *mot des différences finies* $\Delta(w) \in F^*$ de w est défini par

$$\Delta(w) = (w_2 - w_1) \cdot (w_3 - w_2) \cdot (w_n - w_{n-1}),$$

où la soustraction est prise modulo 4.

D'un point de vue géométrique, les lettres **0**, **1**, **2** et **3** correspondent respectivement au mouvement *aller en avant*, *tourner à gauche*, *reculer* et *tourner à droite*. La figure 2.13 illustre l'effet de l'opérateur Δ sur le chemin $w =$ **01012223211**.

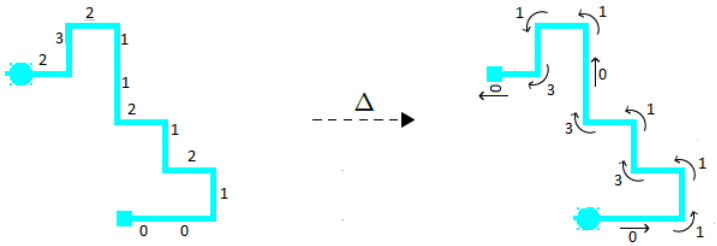


Figure 2.13: Interprétation géométrique de l'opérateur sur le chemin $w =$ 00121211232. Le mot $\Delta(w) =$ 01131301130 décrit les virages effectués pour parcourir le chemin w selon la correspondance suivante : 0 : *aller en avant*, 1 : *tourner à gauche* et 3 : *tourner à droite*. Comme le chemin est auto-évitant, il n'y a pas de lettre 2 : *reculer*.

L'opérateur Δ vérifie la propriété suivante, facile à démontrer :

Proposition 1. *Soient $u = u_1 u_2 ... u_m, v = v_1 v_2 ... v_n \in \mathscr{F}^*$ des mots de longueur m, $n \geq 2$ respectivement. Alors $\Delta(uv) = \Delta(u)\Delta(u_n v_1)\Delta(v)$.*

Afin d'alléger l'écriture de certaines démonstrations, nous étendons la fonction Δ aux couples de mots de la façon suivante. Soient u et v des mots non vides. Alors on écrit $\Delta(u,v) = \Delta(\text{LST}(u)\text{FST}(v))$.

Lorsqu'on considère des chemins non auto-évitants, il est possible que ceux-ci contiennent certains facteurs de l'ensemble $\mathscr{R} = \{\mathbf{02}, \mathbf{13}, \mathbf{20}, \mathbf{31}\}$, c'est-à-dire l'ensemble des chemins de longueur **2** dont les pas sont de directions opposées. Il est alors possible de réduire un mot $w \in \mathscr{F}^*$ en un unique mot w' pour qu'il ne contienne pas de tels facteurs : on supprime alors récursivement toutes les occurrences de facteurs de $\mathscr{R}$ dans w. On remarque par contre que le mot réduit n'est pas forcément auto-évitant.

Exemple 6. *Le mot* $w = \mathbf{0202132031}$ *se réduit à* $w' = \mathbf{00123}$ *après suppression des facteurs* **02**, **13**, **20**, **31**. *Par contre,* w' *n'est pas auto-évitant.*

Exemple 7. *Le mot* $w = \mathbf{01013}$ *est réduit à* $w = \mathbf{010}$ *après suppression du facteur* **13**. *Le mot* $u = \mathbf{11330022}$ *est réduit au mot* **0022** *après la suppression recursive de deux fois le facteur* **13**. *Par la suite* **0022** *est réduit au mot vide* ε *après la suppression récursive de deux fois le facteur* **02**. *Autrement dit, on a les réécritures suivantes :*

$$\mathbf{11330022} \to \mathbf{130022} \to \mathbf{0022} \to \mathbf{02} \to \varepsilon.$$

Il est utile de définir une opération qui inverse en quelque sorte l'effet de l'opérateur Δ. Soit $w = w_1 w_2 ... w_n$ un mot de longueur $n \geq 1$ et $\alpha \in \mathscr{F}$ une lettre. Le *mot des sommes partielles* $\sum_\alpha(w)$de w à partir α est donné par

$$\sum_\alpha(w) = (\alpha) \cdot (\alpha + w_1) \cdot (\alpha + w_1 + w_2) \cdots (\alpha + w_1 + w_2 + ... + w_n).$$

Le *nombre d'enroulements* d'une courbe fermée dans le plan autour d'un point donné est un nombre entier représentant la totalité de fois que la courbe se déplace dans le sens antihoraire autour du point donné. Il dépend donc de l'orientation de la courbe.

La notion de nombre d'enroulements est fondamentale dans l'étude de la topologie algébrique. Elle joue aussi un rôle important en calcul vectoriel, en analyse complexe, en géométrie différentielle et en physique, y compris dans la théorie des cordes. Sur l'alphabet de Freeman, cette notion est très simple à définir et il est également possible de l'étendre aux courbes non fermées. La figure 2.14 illustre deux enroulements autour d'un point p.

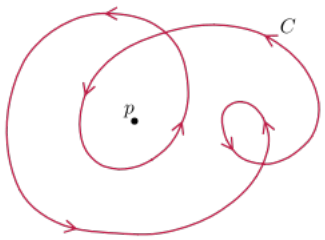


Figure 2.14: Illustration de la notion de nombre d'enroulements : deux enroulements autour du point p

Sur la grille discrète, cela revient à compter les virages à gauche et à droite effectués, alors que les mouvements en avant ne modifient pas le nombre d'enroulements. Les mouvements de recul sont cependant un peu plus complexes à traiter. Plus formellement, soit $w \in \mathscr{F}^*$ un chemin et $w' \in \mathscr{F}^*$ son chemin réduit après suppression récursive des facteurs dans $\{\mathbf{02}, \mathbf{13}, \mathbf{20}, \mathbf{31}\}$. On définit le *nombre d'enroulements* de w par

$$\mathscr{T}(w) = \frac{|\Delta(w')|_1 - |\Delta(w')|_3}{4}.$$

Proposition 2. *Soient* $u = u_1u_2...u_m, v = v_1v_2...v_n \in \mathscr{F}^*$ *des mots de longueur* m, $n \geq 2$ *respectivement. Alors* $\mathscr{T}(uv) = \mathscr{T}(u)\mathscr{T}(u_nv_1)\mathscr{T}(v)$.

Comme pour le mot des différences finies, il est pratique de calculer le nombre d'enroulement

du mot de longueur 2 qui consiste de la dernière lettre de u et de la première lettre de v. Ainsi, on écrit

$$\mathscr{T}(u,v) = \mathscr{T}(\textsc{Lst}(u)\textsc{Fst}(v)).$$

Comme le mot de contour d'un polyomino n'est pas unique, il est pratique de représenter un chemin fermé w par sa classe de conjugaison $[w]$, qu'on appelle aussi *mot circulaire*.

Si w est un chemin fermé, alors un ajustement est nécessaire à la fonction $\mathscr{T}$ puisqu'on prend en compte le virage entre la dernière et la première lettre du mot w. Le *mot des premières différences* d'un mot circulaire $[w]$ est défini par :

$$\Delta([w]) = \Delta([w]) \cdot (w_1 - w_n).$$

Il est possible de réduire un mot circulaire $[w]$ en un mot circulaire $[w']$ en supprimant récursivement les pas de directions opposées de l'ensemble $\{\mathbf{02}, \mathbf{13}, \mathbf{20}, \mathbf{31}\}$. En appliquant la même règle, un mot circulaire $[w]$ est circulairement réduit a un mot unique $[w']$. Il est alors possible d'étendre naturellement la définition de *nombre d'enroulements d'un mot circulaire* par

$$\mathscr{T}([w]) = \tfrac{|\Delta([w'])|_1 - |\Delta([w'])|_3}{4}.$$

Il est bien connu que le nombre d'enroulements d'un mot de contour est contraint :

Théorème 1. ***(Daurat et Nivat, 2003)*** *Soit $[w]$ un mot de contour circulaire. Alors le nombre d'enroulements de $[w]$ est $\mathscr{T}([w]) = 1$ si le parcours est orienté dans le sens anti-horaire et $\mathscr{T}([w]) = -1$ si le parcours est orienté dans le sens horaire.*

La fonction $\mathscr{T}$ ainsi que les isométries ρ^i $(i \in \{0,1,2,3\})$ vérifient les propriétés suivantes :

Proposition 3. *Soit $w \in \mathscr{F}^*$ un chemin.*

1. $\mathscr{T}(\rho^i(w)) = \mathscr{T}(w)$;
2. $\mathscr{T}(\sigma^i(w)) = -\mathscr{T}(w)$;
3. $\mathscr{T}(\widehat{w}) = -\mathscr{T}(w)$.

Démonstration. Soit w un chemin et w' son chemin réduit associé.

1. Le résultat suit du fait que $\Delta(\rho^i(w)) = \Delta(w)$.
2. Il suffit d'observer que la réflexion σ_i inverse les virages à gauche avec les virages à droite et laisse inchangés les pas vers l'avant et les pas vers l'arrière.
3. L'antimorphisme $\widehat{\cdot}$ inverse les lettres **0** et **2** ainsi que les lettres **1** et **3**. Par conséquent

$$\mathscr{T}(\widehat{w}) = \frac{|\Delta(\widehat{w})|_3 - |\Delta([\widehat{w}])|_1}{4} = \frac{|\Delta(w)|_1 - |\Delta([\widehat{w}])|_3}{4} = -\mathscr{T}(w),$$

   tel que voulu.

□

2.5 REPRÉSENTATION DES POLYOMINOS À PARTIR D'UNE IMAGE BINAIRE

Lorsque nous manipulons des régions discrètes, il en existe plusieurs modes de représentations possibles. Pour les polyominos, en plus de la représentation par mot de contour, nous pouvons également utiliser des matrices binaires (voir figure 2.15). Elle présente l'avantage de pouvoir décrire des régions non connexes et avec trou, ce qui est moins naturel dans le cas des mots de contour. Pour un polyomino quelconque, on identifie un pixel par le bit 0 pour signifier qu'il est vide et par le bit 1 pour signifier qu'il est occupé.

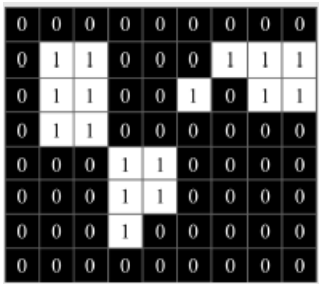


Figure 2.15: Étiquetage de composantes connexes

L'étiquetage en composantes connexes est donc généralement présent lorsqu'il faut regrouper des pixels connexes. Il est à la base de la plupart des chaines algorithmiques en traitement d'images analysant des régions dans une scène. Plusieurs types d'algorithmes standard sont utilisés pour coder l'information et leur efficacité varie suivant la nature des données. On trouvera ci-dessous une description sommaire de quelques techniques courantes.

L'approche pixel est une approche de comparaison de l'étiquette courante avec les étiquettes du voisinage (voir figure 2.16).

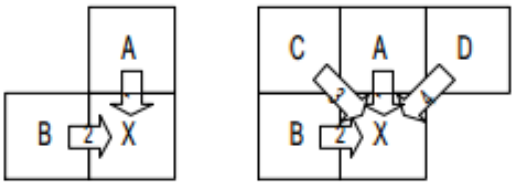


Figure 2.16: Étiquetage 4-connexe et 8-connexe

Le codage par arbre quaternaire est une méthode de représentation permettant de minimiser l'information nécessaire pour coder des zones homogènes. C'est une forme de compactage qui divise l'image en quatre quadrants (voir figure 2.17). Chaque quadrant est à son tour divisé en 4 sous quadrants et ainsi de suite.

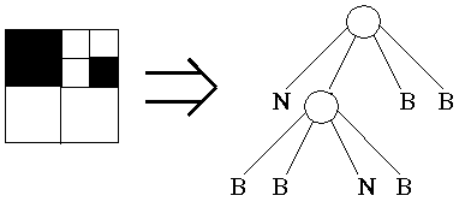


Figure 2.17: Arbre quatrenaire

En balayant l'image ligne par ligne, il est possible d'appliquer l'encodage de plage sur toutes les séquences de points qui ont la même couleur. Le codage par longueur de plage (RLC) est utilisé pour rechercher des segments adjacents pour déterminer la connectivité des segments. Pour réduire la complexité algorithmique et le rendre plus compacte, le codage par arbre quaternaire est généralement associé à RLC.

Cependant ces approches utilisent un grand nombre de tests. Ce qui se traduit au niveau du processeur par une importante consommation de ressource [Lacassagne et al., 1998].

Nous proposons de représenter de façon compacte une image par une matrice binaire, à l'aide d'une liste d'entiers, où chaque ligne (ou, de façon équivalente, chaque colonne) correspond à la représentation binaire de l'entier en question (voir figure 2.18). Par exemple, pour des tétraminos, on se limite à des entiers (voir figure 2.19) allant de 1 à 15.

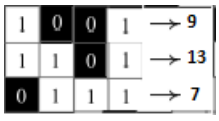


Figure 2.18: Pour ce polyomino (1 représentant une case occupée, le 0 représentant une case vide), chaque ligne correspond donc respectivement aux entiers 9, 13 et 7.

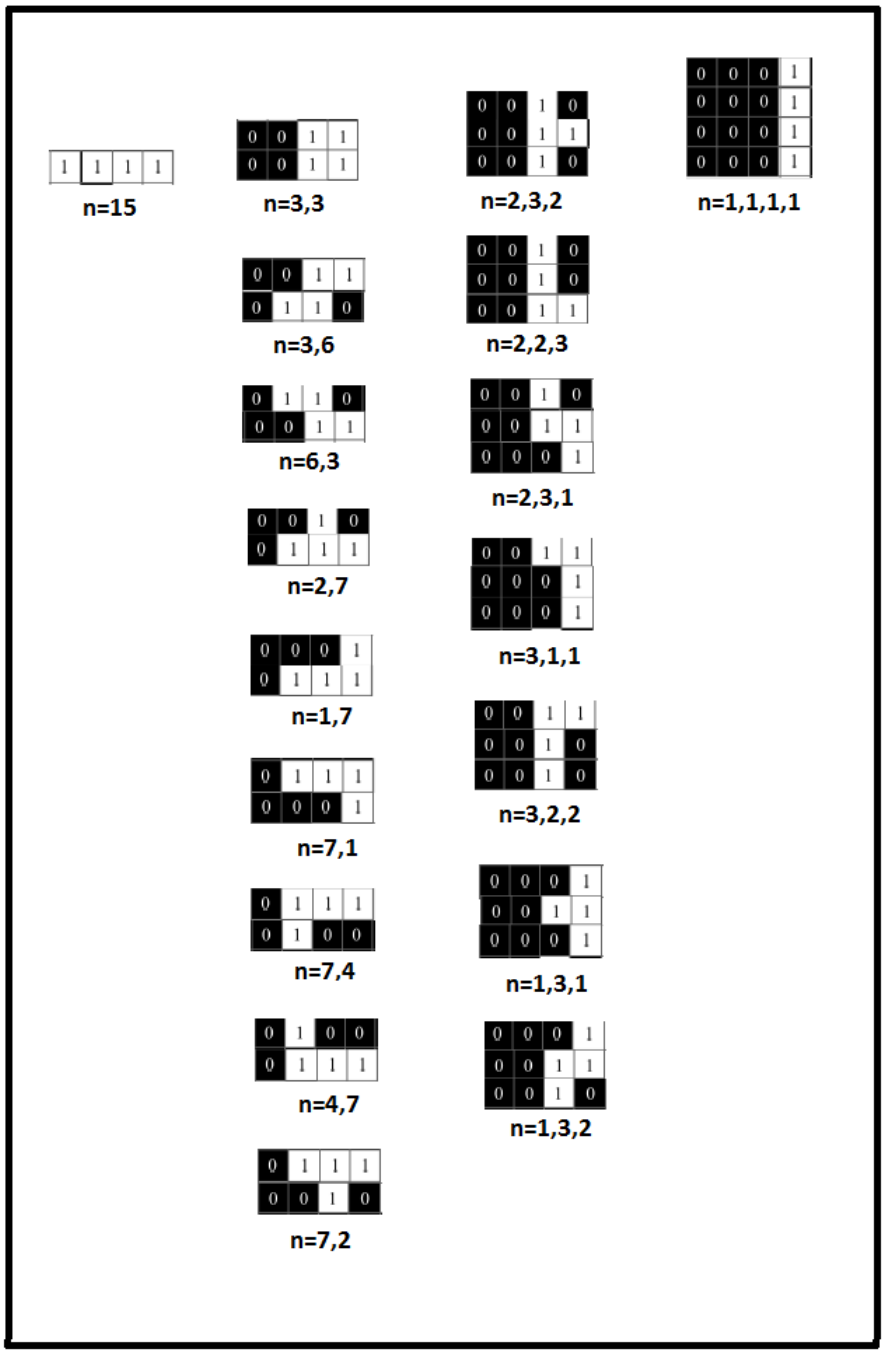


Figure 2.19: Tous les polyominos de 4 cellules

CHAPITRE 3

ARITHMÉTIQUE DES POLYOMINOS

Dans ce chapitre, nous nous intéressons à une classe de morphismes appelés *morphismes homologues* pour définir la notion de polyominos premiers et composés. Nous étudions par la suite les aspects algorithmiques des ces objets.

3.1 POLYOMINOS PARALLÉLOGRAMMES

Dans [Beauquier et Nivat, 1991], **D. Beauquier** et **M. Nivat** démontrent qu'un polyomino P pave un plan par translation si et seulement s'il admet un mot de contour

$$w = xyz\widehat{xyz},$$

avec au plus un mot vide ε parmi x, y ou z et où l'antimorphisme $\widehat{\ }$ est défini au chapitre précédent (page 19). La factorisation $(xyz\widehat{xyz})$ est appelée une *BN-factorisation* de P et, par abus de notation, on écrit simplement $xyz\widehat{xyz}$. Si x, y ou z est un mot vide alors P est un polyomino parallélogramme (**tuile carrée** dans [Brlek et Provençal, 2006]). Autrement, on parle de polyomino hexagone (**tuile hexagone** dans [Brlek et Provençal, 2006]). Il est à noter que les **polyominos parallélogrammes** (voir figure 3.1) ont été introduits, il y a plus de 40 ans,

avec un sens totalement différent de celui utilisé dans le mémoire [Polyá, 1969]. Néanmoins, nous préférons garder la terminologie **parallélogramme** puisque qu'elle semble appropriée.

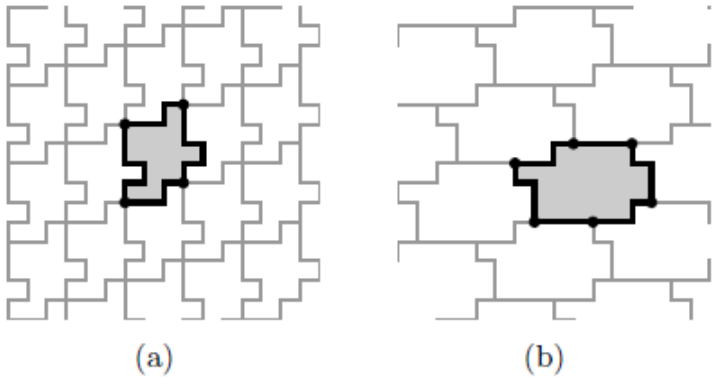


Figure 3.1: (a) Un polyomino parallélogramme et le pavage qu'il induit (le mot de contour admet une BN-factorisation $0010 \cdot 101211 \cdot 2322 \cdot 330323$). Les points noirs indiquent les points de factorisation du mot de contour et forment clairement un parallélogramme. (b) Un polyomino hexagone ayant comme mot de contour $000 \cdot 0010 \cdot 1121 \cdot 222 \cdot 2322 \cdot 3033$ et le pavage correspondant.

Proposition 4. *[Brlek et al., 2005] Le nombre d'enroulements d'un mot de contour w est $\mathscr{T}([w]) \pm 1$.*

Une notion particulièrement utile est celle d'*orientation de chemin*. On dit que w est *positivement orienté* si son nombre d'enroulement est $\mathscr{T}([w]) = 1$ (autrement dit, on le parcourt dans le sens anti-horaire). En général, si $XY\widehat{X}\widehat{Y}$ est un chemin (simple ou complexe), son nombre d'enroulement peut être calculé à partir de sa BN-factorisation par la formule :

$$\mathscr{T}([XY\widehat{X}\widehat{Y}]) = \mathscr{T}(X,Y) + \mathscr{T}(Y,\widehat{X}) + \mathscr{T}(\widehat{X},\widehat{Y}) + \mathscr{T}(\widehat{Y},X).$$

Par conséquent, chaque tuile carrée satisfait la condition suivante :

Proposition 5. *Soit $w = XY\widehat{X}\widehat{Y}$ le mot de contour d'une tuile carrée alors :*

$$\Delta(X,Y) = \Delta(Y,\widehat{X}) = \Delta(\widehat{X},\widehat{Y}) = \Delta(\widehat{Y},X) = \alpha,$$

où le parcours est orienté positivement si $\alpha = \mathbf{1}$ et négativement si $\alpha = \mathbf{3}$.

3.2 POLYOMINOS PREMIERS ET COMPOSÉS

Définition 3. *[Blondin Massé et al., 2014] Un morphisme $\varphi : \mathscr{F}^* \to \mathscr{F}^*$ est appelé* homologue *si $\varphi(a) = \widehat{\varphi(\overline{a})}$ pour tout $a \in \mathscr{F}$.*

Autrement dit, les morphismes homologues remplacent les deux pas élémentaires horizontaux et verticaux par des chemins arbitraires. En effet, pour l'étape horizontale, $\varphi(\mathbf{0})$ est traversé en direction contraire par rapport à $\varphi(\mathbf{2})$. La même idée s'applique pour l'étape verticale. En effet, $\varphi(\mathbf{1})$ est traversé en direction contraire par rapport à $\varphi(\mathbf{3})$.

Exemple 8. *Considérons le morphisme $\varphi : \mathscr{F}^* \to \mathscr{F}^*$ défini par*

$$\varphi(\mathbf{0}) = \mathbf{01000}, \quad \varphi(\mathbf{1}) = \mathbf{121}, \quad \varphi(\mathbf{2}) = \mathbf{22232} \quad \textit{et} \quad \varphi(\mathbf{3}) = \mathbf{303}.$$

On a

$$\begin{aligned} \widehat{\varphi(\mathbf{0})} &= \widehat{\mathbf{01000}} = \mathbf{22232} = \varphi(\mathbf{2}), \\ \widehat{\varphi(\mathbf{1})} &= \widehat{\mathbf{121}} = \mathbf{303} = \varphi(\mathbf{3}), \end{aligned}$$

de sorte que φ est un morphisme homologue.

Une conséquence immédiate de la définition est que tout morphisme homologue commute avec l'antimorphisme $\widehat{\ }$:

Proposition 6. *Pour tout morphisme homologue φ et pour tout $w \in \mathscr{F}^*$, $\widehat{\varphi(w)} = \varphi(\widehat{w})$.*

Démonstration. Soit $n = |w|$ et $w = w_0 w_1 \cdots w_n$. Nous avons

$$\begin{aligned}
\widehat{\varphi(w)} &= \varphi(w_0 \widehat{w_1 \cdots w_{n-1}} w_n) \\
&= \varphi(w_0)\varphi(w_1)\widehat{\cdots}\widehat{\varphi(w_{n-1})}\varphi(w_n) \\
&= \widehat{\varphi(w_n)}\widehat{\varphi(w_{n-1})}\cdots\widehat{\varphi(w_2)}\widehat{\varphi(w_1)} \quad \text{(propriété d'un antimorphisme)} \\
&= \varphi(\overline{w_n})\varphi(\overline{w_{n-1}})\cdots\varphi(\overline{w_2})\varphi(\overline{w_1}) \quad \text{(par définition de morphisme homologue)} \\
&= \varphi(\overline{w_n w_{n-1} \cdots w_2 w_1}) \\
&= \varphi(\widehat{w}),
\end{aligned}$$

tel que voulu. □

Comme les polyominos sans trou ont un mot de contour simple, une condition additionnelle des morphismes homologues est nécessaire afin de définir les polyominos sans trou premiers et composés.

Définition 4. *Soit φ un morphisme homologue. Nous disons que φ est un* morphisme parallélogramme *si :*

1. *$\varphi(\mathbf{0123})$ est un mot de contour, c'est-à-dire un chemin fermé et auto-évitant ;*
2. *$\textsc{Fst}(\varphi(a)) = a$ pour tout $a \in \mathscr{F}$.*

Exemple 9. *Le morphisme φ défini par*

$$\varphi(\mathbf{0}) = \mathbf{00}, \quad \varphi(\mathbf{1}) = \mathbf{121}, \quad \varphi(\mathbf{2}) = \mathbf{22} \quad \textit{et} \quad \varphi(\mathbf{3}) = \mathbf{303}$$

est un morphisme parallélogramme. En effet, le chemin $\varphi(\mathbf{0123})$ est bien un mot de contour (voir figure 3.2).

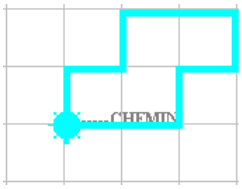


Figure 3.2: La tuile induite par le morphisme parallélogramme φ défini par $0 \mapsto \mathbf{00}, \mathbf{1} \mapsto \mathbf{121}, \mathbf{2} \mapsto \mathbf{22}, \mathbf{3} \mapsto \mathbf{303}$.

La définition de morphisme parallélogramme induit des contraintes sur les premières et dernières lettres de la façon suivante :

Proposition 7. *Soit φ un morphisme parallélogramme. Alors*

1. $\textsc{Fst}(\varphi(\mathbf{0})) = \textsc{Lst}(\varphi(\mathbf{0}))$ *et* $\textsc{Fst}(\varphi(\mathbf{1})) = \textsc{Lst}(\varphi(\mathbf{1}))$ *;*
2. *Les premières lettres sont mutuellement distinctes, c'est-à-dire*

$$\{\textsc{Fst}(\varphi(\mathbf{0})), \textsc{Fst}(\varphi(\mathbf{1})), \textsc{Fst}(\varphi(\mathbf{2})), \textsc{Fst}(\varphi(\mathbf{3}))\} = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}.$$

Démonstration. Notons que $\varphi(\mathbf{0123})$ décrit une tuile parallélogramme dans le sens positif. Par la proposition 5, nous avons donc $\textsc{Lst}(\varphi(\mathbf{0})) + \mathbf{1} = \textsc{Fst}(\varphi(\mathbf{1}))$. Similairement, $\textsc{Lst}(\varphi(\mathbf{1})) + \mathbf{1} = \textsc{Fst}(\varphi(\mathbf{2}))$. De plus, on sait que $\textsc{Fst}(\varphi(\mathbf{2})) = \textsc{Lst}(\varphi(\mathbf{0})) + \mathbf{2}$, puisque $\varphi(\mathbf{2}) = \widehat{\varphi(\mathbf{0})}$. En soustrayant $\mathbf{1}$ de chaque côté de ces deux dernières équations, nous obtenons $\textsc{Lst}(\varphi(\mathbf{1})) = \textsc{Lst}(\varphi(\mathbf{0})) + \mathbf{1}$, d'où $\textsc{Lst}(\varphi(\mathbf{1})) = \textsc{Fst}(\varphi(\mathbf{1}))$. De la même manière, nous obtenons $\textsc{Lst}(\varphi(\mathbf{0})) = \textsc{Fst}(\varphi(\mathbf{0}))$. À partir de ces équations, il s'ensuit que l'ensemble des premières lettres est bien $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\}$. □

La proposition 7, démontrée dans [Blondin Massé et al., 2013], est l'idée principale sur laquelle nous nous appuyons pour la conception des algorithmes de factorisation de polyominos et l'étude de leur primalité. Ce fait indique en effet que les facteurs d'un polyomino parallélogramme quelconque commencent et se terminent par le même pas élémentaire.

Les morphismes parallélogrammes et homologues forment des sous-monoïdes du monoïde des morphismes sur $\mathscr{F}$:

Proposition 8. *Soit $\mathscr{M}$ l'ensemble des morphismes de $\mathscr{F}$, $\mathscr{H}$ l'ensemble des morphismes homologues et $\mathscr{P}$ l'ensemble des morphismes parallélogrammes. Alors $\mathscr{H}$ et $\mathscr{P}$ sont des sous-monoïdes de $\mathscr{M}$ par rapport à la composition.*

Démonstration. Tout d'abord, l'élément neutre est le morphisme identité Id qui est à la fois un morphisme homologue et parallélogramme.

Soit $\varphi, \varphi' \in \mathscr{H}$. Alors pour chaque pas élémentaire $a \in \mathscr{F}$:

$$\widehat{(\varphi \circ \varphi')}(\overline{a}) = \widehat{\varphi(\varphi'(\overline{a}))} = \varphi(\widehat{\varphi'(\overline{a})}) = \varphi(\varphi'(a)) = (\varphi \circ \varphi')(a),$$

de sorte que $\varphi \circ \varphi' \in \mathscr{H}$. Similairement si $\varphi, \varphi' \in \mathscr{P}$, alors

$$\textsc{Fst}((\varphi \circ \varphi')(a)) = \textsc{Fst}((\varphi(\varphi'(a))) = \textsc{Fst}(\varphi'(a)) = a.$$

Il est clair que $(\varphi \circ \varphi')(\mathbf{0123})$ est fermé. En outre, on montre que $(\varphi \circ \varphi')(\mathbf{0123})$ est auto-évitant, ce qui termine la démonstration. □

Chaque morphisme parallélogramme φ est associé à un polyomino unique, dénoté par $\textsc{Poly}(\varphi)$. Inversement, on pourrait être tenté de dire que chaque polyomino unique, dénoté par $\textsc{Poly}(\varphi)$ est uniquement représenté par un morphisme parallélogramme. À cet effet, la condition (i) de la définition 2 assure que le mot de contour est traversé dans le sens horaire et que seulement un de ses conjugués est choisi. Cependant, même en tenant ces restrictions en compte, il existe des polyominos parallélogrammes avec deux factorisations parallélogrammes distinctes.

Exemple 10. *Les morphismes parallélogrammes* φ *définis par* $\varphi(\mathbf{0}) = \mathbf{010}, \varphi(\mathbf{1}) = \mathbf{121}$ *et* $\varphi'(\mathbf{0}) = \mathbf{030}, \varphi'(\mathbf{1}) = \mathbf{101}$*, produisent l'un et l'autre le pentamino X (voir figure 3.3).*

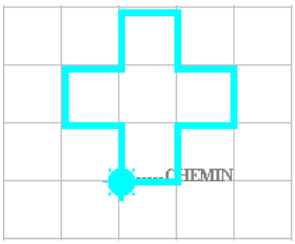


Figure 3.3: Pentamino X

Les tuiles admettant deux factorisations distinctes non triviales sont appelées **doubles parallélogrammes** et ont été décrites dans [Blondin Massé et al., 2013]. De plus, il n'existe aucun polyomino parallélogramme décrit par plus de deux morphismes [Blondin Massé et al., 2012].

Nous sommes maintenant prêts à définir les polyominos premiers et composés. Il devrait être noté que nous présentons une définition équivalente à celle donnée dans [Provençal, 2008] qui repose sur la notion de morphisme parallélogramme.

Définition 5. *Soit* Q *un polyomino quelconque et* φ *un morphisme parallélogramme. Alors la* composition de φ et Q*, dénotée par* $\varphi(Q)$*, est le polyomino unique de mot de contour* $\varphi(u)$*, où* u *est un mot de contour de* Q*.*

Des exemples sont présentés dans les figures 3.4, 3.9, 3.10 et 3.11.

Définition 6. *Un polyomino non unitaire* P *est dit* composé *s'il existe un mot de contour* u *et un morphisme parallélogramme* φ *tel que :*

1. $\textsc{Poly}(\varphi(u)) = P$;
2. $\textsc{Poly}(u)$ n'est pas le carré unitaire ;

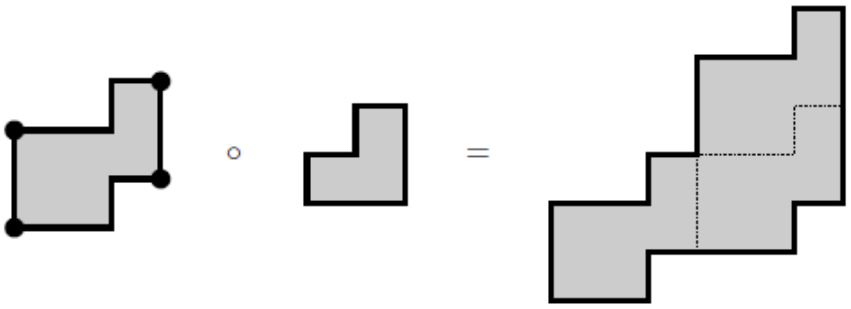


Figure 3.4: Construction d'un polyomino avec un polyomino parallélogramme. Le morphisme parallélogramme est illustré avec des points noirs. Le polyomino composé résultant peut être pavé par des copies du polyomino parallélogramme.

3. $\varphi \neq \mathrm{Id}$.

Sinon , P est dit *premier*.

En d'autres termes, un polyomino est premier si et seulement s'il ne peut pas être décomposé non trivialement comme le produit d'un morphisme parallélogramme et d'un autre polyomino. Il est utile de noter que le problème de décider si un polyomino est premier est au moins aussi difficile que le problème de décider si un nombre est premier. En effet, on peut remarquer que le rectangle $1 \times n$ ayant le mot de contour $\mathbf{0}^n\mathbf{12}^n\mathbf{3}$ est premier si et seulement si n est premier. Dans le même esprit, un mot de contour u est appelé *premier* si $\textsc{Poly}(u)$ et un polyomino premier et un morphisme parallélogramme φ est appelé *premier* si $\textsc{Poly}(\mathbf{0123})$ est un polyomino premier.

3.3 THÉORÈME DE DÉCOMPOSITION EN FACTEURS PREMIERS

3.3.1 GÉNÉRATION ET FACTORISATION DE NOMBRES PREMIERS

Dans la cryptographie traditionnelle, en vigueur avant les années 1970, les opérations de chiffrement et de déchiffrement étaient effectuées avec une même clé unique (voir figure 3.5).

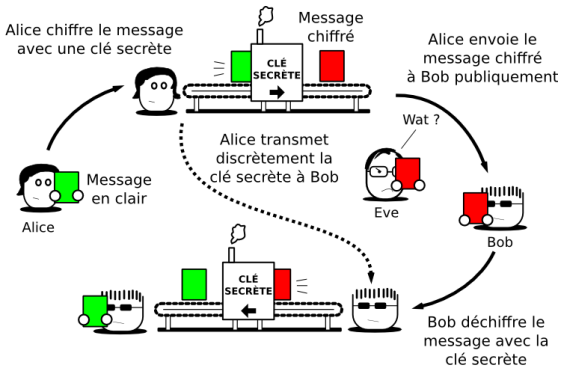


Figure 3.5: Chiffrement symétrique : La clé secrète qui chiffre est la clé secrète qui déchiffre `<http://www.cyphercat.eu/expl_chiffrement_symetrique.php>`**.**

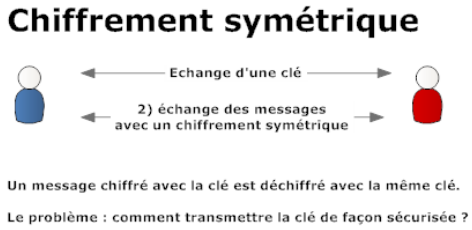


Figure 3.6: Lacune du chiffrement symétrique `<http://www.kitpages.fr/fr/cms/93/chiffrements-symetriques-et-asymetriques>`

Cette façon restrictive d'opérer a incité deux chercheurs de l'université Stanford, Whitfield Diffie et Martin Hellman, à écrire en 1976 un célèbre article [Diffie et Hellman, 1976], dans lequel ils suggèrent que peut-être le chiffrement et déchiffrement pourraient être faits avec une paire de clés différentes plutôt qu'avec la même clé. La clé de déchiffrement serait alors

gardée secrète comme étant la paire de clé privée (voir figure 3.7). La clé de chiffrement est publique sans compromettre la sécurité du déchiffrement.

La *théorie des nombres* s'avère utile en ce qui concerne la sécurité informatique. Le concept a été appelé la cryptographie à clé publique ou cryptographie asymétrique et repose sur la théorie des nombres[Rivest et al., 1978]. Elle permet de protéger les données sensibles telles que les numéros de carte de crédit.

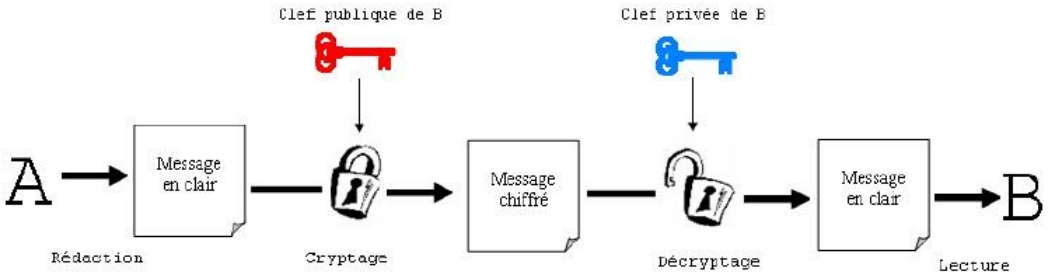


Figure 3.7: ***Les fonctions à sens uniques*** **sont des fonctions mathématiques telles qu'une fois appliquées à un message, il est extrêmement difficile de retrouver le message original. L'existence d'une** ***brèche secrète*** **permet cependant à une personne de décoder facilement le message grâce à un élément d'information. Les fonctions à sens unique avec brèche secrète sont à la base de RSA. La fonction à sens unique à brèche secrète est la « clé publique ». L'élément d'information est la « clé privée ». Ce cryptosystème à clé publique utilise effectivement deux clés : une publique et une privée. Celles-ci sont générées à partir des nombres premiers. Pour transmettre un message à Bob, Alice utilise la clé publique de Bob. Seule la clé privée secrète de Bob peut déchiffrer le message.** `<http://mbourgeois.developpez.com/articles/securite/pgp/>`**.**

La cryptographie asymétrique est donc basée sur la théorie des nombres premiers, et sa robustesse tient du fait qu'il n'existe aucun algorithme efficace de décomposition d'un nombre en facteurs premiers. Nous notons les observations suivantes :

Observation 2. *La génération de nombres premiers est simple : Il est facile de trouver un nombre premier aléatoire d'une grandeur donnée. Pour générer un nombre premier aléatoire,*

on peut simplement générer des nombres aléatoires de longueurs données et faire des tests de primalité sur ces nombres jusqu'à ce qu'un nombre premier soit trouvé.

Observation 3. *La multiplication est simple : compte tenu deux grands nombres premiers p et q, il est facile de calculer leur produit, $N = pq$ (complexité quadratique).*

Observation 4. *La factorisation est difficile : le problème consistant à retrouver p et q connaissant N est difficile quand p et q sont grands. C'est le problème de factorisation (voir Figure 3.8). Pour preuve de la difficulté de ce problème, l'algorithme le plus efficace à ce jour, connu sous le nom de crible a une complexité super-polynomiale [Lenstra et al., 1993].*

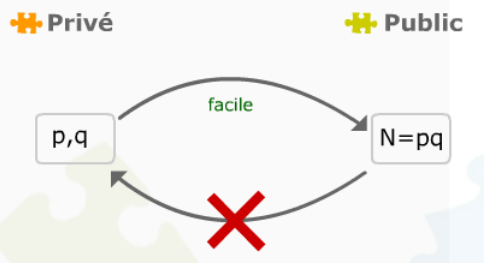


Figure 3.8: Une *primitive* est construite à l'aide de deux grands nombres premiers p et q distincts et de tailles similaires. Le problème de la factorisation ci-dessus est la primitive RSA.

Il n'est pas toujours facile de déterminer si un nombre est premier. En fait, il semblerait que pour tester la primalité d'un nombre, on aurait besoin de déterminer tous les facteurs du nombre pour voir s'il existe un diviseur autre que le nombre en question et 1. Des méthodes plus rapides pour les tests de primalité découvertes dans les années 1970 permettent de déterminer des nombres premiers en exploitant certaines de leurs propriétés [Miller, 1976]. Sans ces résultats de détection de nombres premiers, beaucoup de clés publiques cryptographiques actuelles ne seraient pas pratiques voire obsolètes du fait que les méthodes efficaces de génération de grands nombres premiers permettent de construire de longues paires de clés publiques/privées.

Conséquemment aux longues paires de clés publiques/privées, les méthodes de calcul actuels pour déterminer si un grand nombre est premier sont lentes et coûteuses. Par exemple, il a

été récemment estimé que la récupération des facteurs premiers d'un nombre de 1024 bits prendrait une année sur une machine coûtant 10 millions de dollars US [Franke et al., 2005]. Un nombre de 2048 bits nécessiterait plusieurs billiards de fois plus de travail. Ces estimations actuelles sont ce à quoi on aurait pu s'attendre dans les années 1970 lorsque le problème a été proposé [Rivest et al., 1978]. Parallèlement, la taille recommandée des clés de chiffrement devient de plus en plus importante en raison d'une puissance de calcul en croissance constante, soit de plus en plus d'outils puissants de cryptanalyse. Ainsi, personne ne sait si on pourra découvrir des longueurs de clés plus importantes dans les années à venir.

3.3.2 THÉORÈME FONDAMENTAL DE L'ARITHMÉTIQUE

S'il existe un algorithme simple à mettre en place pour décomposer un nombre de taille raisonnable, cet algorithme se révèle rapidement inefficace, en termes de temps, pour de très grands nombres. La recherche d'algorithmes performants est donc très pertinente. En outre, si une méthode rapide était trouvée pour résoudre le problème de la factorisation des nombres entiers, alors plusieurs systèmes cryptologiques importants seraient cassés, incluant l'algorithme à clé publique RSA.

Après la définition 6, on est naturellement amené à se demander si le théorème fondamental de l'arithmétique peut être étendu aux polyominos. Il y a deux conditions qui doivent être vérifiées : l'**existence** d'une factorisation en nombres premiers et l'**unicité** de cette factorisation.

La première est facile à prouver :

Théorème 5. *Soit P un polyomino non unitaire sans trou. Il y a deux cas possibles :*

1. *P est un polyomino premier ;*
2. *Il existe des morphismes parallélogrammes premiers* $\varphi_1, \varphi_2, \ldots, \varphi_n$ *et un mot de contour*

premier u tel que

$$P = \text{POLY}((\varphi_1 \circ \varphi_2 \circ \ldots \circ \varphi_n)(u)).$$

Démonstration. Par induction sur le périmètre de P. Si P est premier, alors il n'y a rien à prouver. Sinon, il existe un morphisme parallélogramme $\varphi \neq \text{Id}$ et un mot de contour u, avec $\text{POLY}(u)$ différent du carré unité, tel que $P = \text{POLY}(\varphi(u))$. Par induction appliquée à $\text{POLY}(u)$, il existe des morphismes parallélogrammes φ_1, φ_2, …, φ_n et un mot de contour premier v tel que

$$u = (\varphi_1 \circ \varphi_2 \circ \ldots \circ \varphi_n)(v),$$

de sorte que $P = \text{POLY}((\varphi \circ \varphi_1 \circ \varphi_2 \circ \ldots \circ \varphi_n)(v))$. Si φ est premier alors l'affirmation est prouvée. Sinon, en utilisant l'induction, on montre que φ est égal à $\varphi'_1 \circ \varphi'_2 \circ \ldots \circ \varphi'_k$ pour des parallélogramme premiers φ'_1, φ'_2,…, φ'_k. Ce qui conclut la preuve sur l'existence d'une factorisation première. □

Le théorème 5 garantit l'existence d'une factorisation en polyominos premiers. Cependant, il semble plus difficile de démontrer si une telle factorisation est unique. D'un point de vue expérimental, nos explorations informatiques faites avec des polyominos de périmètre ne dépassant pas **900** n'ont pour le moment pas produit de contre-exemple. Nous sommes donc mené à proposer la conjecture qui suit :

Conjecture 6. *Soit P un polyomino composé quelconque. Alors, il existe un unique morphisme parallélogramme* φ_1 *et un unique mot de contour u premier tel que* $P = \text{POLY}((\varphi_1 \circ (u)))$

3.4 ALGORITHMES DE FACTORISATION

Afin de déterminer si un polyomino P est composé, il suffit de trouver un morphisme parallélogramme $\varphi \neq \mathrm{Id}$ et un mot de contour $u \notin [\mathbf{0123}]$ tel que $\varphi(u)$ soit le mot de contour de P. Si un tel morphisme est inexistant, alors nous pouvons en conclure que le polyomino est premier.

3.4.1 VERSION NAÏVE

L'approche directe et simple consiste à essayer chaque factorisation possible jusqu'à ce qu'on trouve un morphisme parallélogramme capable de factoriser le mot de contour du polyomino. Cette approche brute de stockage d'occurrences, bien que coûteuse, présente un niveau raisonnable d'efficacité et nous garantit entre autres de couvrir tous les morphismes homologues existants appliqués au pas élémentaire si celui-ci est composé.

Pour réduire le nombre de cas à prendre en compte, on se limite à des facteurs commençant et se terminant par la même lettre. Plus précisément, soit P un polyomino et w l'un de ses mots de contour, pour tout $a \in \mathscr{F}$, soit F_a l'ensemble des facteurs de w^2 commençant et se terminant par a. Les étapes ci-dessous peuvent être utilisées pour factoriser P :

1. Calculer les occurrences $u \in F_{\mathbf{0}}$, $v \in F_{\mathbf{1}}$;
2. Soit $u \in F_{\mathbf{0}}$, $v \in F_{\mathbf{1}}$;
3. Soit φ le morphisme parallélogramme induit par u et v ;
4. S'il y a un conjugué quelconque w' de w tel que $w = \varphi(w')$, alors retourner (φ, u).
5. Autrement, répéter les étapes 2–4 jusqu'à ce que tous les u et v possibles soient épuisés.

La complexité de l'algorithme ci-dessus est clairement polynomiale.

Théorème 7. *Tout polyomino P peut-être factorisé en produit de polyominos premiers en $\mathscr{O}(n^7)$ où n est le périmètre de P.*

Démonstration. Soit w tout mot de contour de P. L'étape 1 est realisée en $\mathscr{O}(n^2)$ puisqu'il y a $\mathscr{O}(n^2)$ facteurs commençant et se terminant par le même pas élémentaire dans tout mot de $\mathscr{F}$. Donc, il y a au plus n^2 valeurs possibles. Les étapes $2-4$ sont alors répetées en $\mathscr{O}(n^4)$ puisqu'il y a au plus n^2 comparaisons possibles considérant les chemins horizontaux et verticaux. La construction de φ à l'étape 3 est alors faite en temps constant alors que l'étape 4 prend $\mathscr{O}(n)$ puisque celle-ci doit être effectuée pour chaque conjugué de w. En conséquence, décomposer P par $\varphi(u)$, pour un quelconque morphisme parallélogramme φ et un mot de contour u est fait en $\mathscr{O}(n^5)$. Finalement, quand la liste des $\varphi(u)$ est formée pour chaque $a \in \mathscr{F}$, il reste à vérifier si w est décodé par φ, en fissurant w, considérant ainsi la longueur du morphisme homologue pour les chemins horizontaux et verticaux. Cette opération est faite au plus en temps linéaire. Dès lors que cela doit être répété aussi longtemps que φ ou u n'est pas premier, la complexité globale est $\mathscr{O}(n^7)$.

Par conséquent : □

Corollaire 1. *Étant un polyomino P de périmètre n, il peut être décidé en temps polynomial par rapport à n si P est premier ou composé.*

Algorithme 1 : Algorithme naïf (occurrences, principale)

```
1  Données : w est fermé, mot de freeman, simple et auto − évitant ;
2  Fonction OCCURENCES(w : mot de contour, a ∈ 𝔉)
3  début
       Sorties : F_a
4      x = 0
5      pour i ← 1 à |w| faire
6          pour j ← i à |w| faire
7              si (w[i] = a et w[j] = a et i ≤ j) alors
8                  occurence ← w[i : j]
9                  occurence ∈ F_a
10                 x++
11     retourner F_a
12 Fonction principale(w : mot de contour, a ∈ 𝔉)
13 début
       Sorties : φ
14     F_0 = OCCURENCES (w, 0)
15     F_1 = OCCURENCES (w, 1)
16     pour i ← 1 à |F_0| faire
17         pour j ← 1 à |F_1| faire
18             u = F_0(i)
19             v = F_1(j)
20             φ(0) = u
21             φ(1) = v
22             factorisation=faux
23             FACTORISATION(w, φ(0), φ(1))
24             si (factorisation==vrai) alors
25                 retourner φ /* Le morphisme parallélogramme est complètement défini
                       et non trivial, on le retourne. Donc w' = φ(w) est un mot de
                       contour fermé et auto-évitant. On a alors une figure discrète
                       composée                                                    */
26             sinon
27                 retourner ∅ /* Si φ est un ensemble vide alors on peut conclure
                       qu'on a alors une figure discrète première                   */
```

Algorithme 2 : Algorithme naïf (Factorisation)

```
1  Fonction FACTORISATION(w : mot de contour, φ(0), φ(1))/* On considère ici les
       caractéristiques de la longueur de φ(0) et φ(1) pour factoriser w.        */
2  début
       Entrées : mot de contour, dimension x, dimension y
3      /* x = |φ(0)|                                                            */
4      /* y = |φ(1)|                                                            */
5      pour i ← 1 à |w| faire
6          y=0
7          si w_i == 0 ou w_i == 2 alors
8              scission[y] = w[i...x], i = x, x = +i
9          si w_i == 1 ou w_i == 3 alors
10             scission[y] = w[i...y], i = y, y = +i
11     pour i ← 1 à |scission| faire
12         w' ← scission[i][premier indice] /* On récupère le premier indice de chaque
                découpage et on s'assure que w' est fermé et auto-évitant. Si oui
                w = φ(w').                                                       */
13     si w' est fermé, simple et auto − évitant == vrai alors
14         polyomino est composé
15         retourner w'
16         factorisation ←vrai
17     si factorisation==faux alors
18         polyomino est premier
```

L'algorithme a été implémenté en Java et testé sur des polyominos premiers et composés. Les détails de l'évaluation empirique sont donnés dans le chapitre 4.

3.4.2 AMÉLIORATION DE L'ALGORITHME NAÏF

Une complexité de $\mathscr{O}(n^7)$ est plutôt grande et il est raisonnable d'espérer la réduire. Plus spécifiquement, au lieu d'énumérer tous les facteurs possibles u dans $\mathscr{F}_{\mathbf{0}}$ et v dans $\mathscr{F}_{\mathbf{1}}$, le choix devrait plutôt s'opérer selon que u et v surviennent de façon contiguë dans le mot en traitement. Dès lors, cela donne l'approche améliorée suivante.

Comme première étape, nous choisissons n'importe quel mot de contour w commençant par $\mathbf{0}$. La première idée consiste à essayer de diviser w par des blocs commençant et se terminant par le même pas élémentaire, conformément à la proposition 2. Il suffit donc de regarder toutes les occurrences de $\mathbf{0}$ pour déterminer un $\varphi(\mathbf{0})$. Quand un tel bloc est choisi, alors $\varphi(\mathbf{0})$ et $\varphi(\mathbf{2})$ sont complètement déterminés.

L'étape suivante consiste à vérifier le pas élémentaire suivant le premier bloc. Si celui-ci est $\mathbf{0}$ ou $\mathbf{2}$ alors nous vérifions si les lettres suivantes ou le bloc suivant correspondent à $\varphi(\mathbf{0})$. Si tel n'est pas le cas, alors nous devons choisir une autre valeur $\varphi(\mathbf{0})$. D'autre part, s'il y a une correspondance, alors on passe au bloc suivant. Nous répétons les étapes précédentes jusqu'à ce que nous atteignions la lettre $\mathbf{1}$ ou $\mathbf{3}$. De la même manière que pour la lettre $\mathbf{0}$, nous essayons ensuite chaque bloc possible pour chaque $\mathbf{1}$ ou $\mathbf{3}$. Quand un tel bloc est choisi, alors $\varphi(\mathbf{1})$ et $\varphi(\mathbf{3})$ sont complètement déterminés. Quand les quatre images de φ sont choisies, il ne reste plus qu'à vérifier si le mot de contour peut être factorisé comme un produit des quatre blocs (cette étape est appelée étape de décodage).

Cela nous mène naturellement à l'algorithme 3 qui permet de factoriser tout polyomino

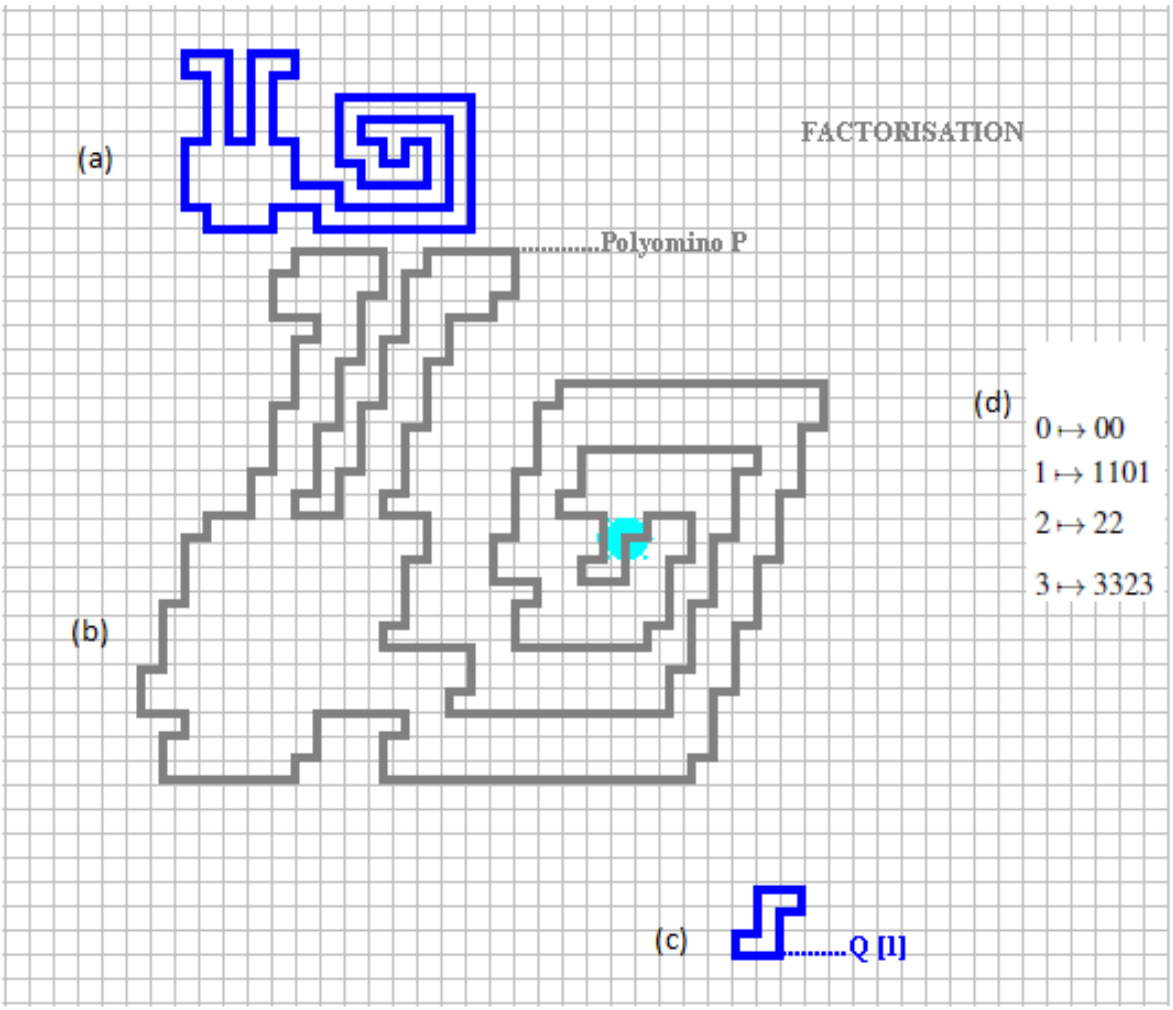


Figure 3.9: Illustration du travail de notre algorithme de factorisation de polyominos (implémentation en Java en utilisant le module Turtle de Python). Soit la factorisation d'une figure discrète composée en forme d'escargot (Figure a) de mot de contour 332322...00 et de longueur du mot de contour $L = 268$, dans le sens antihoraire. L'algorithme détecte la figure a comme étant l'application du morphisme φ (Figure d), representé par le morphisme parallélogramme Q (Figure c), appliqué au contour de la figure discrète première P en forme d'escargot (Figure b). Le tout en quelques millisecondes.

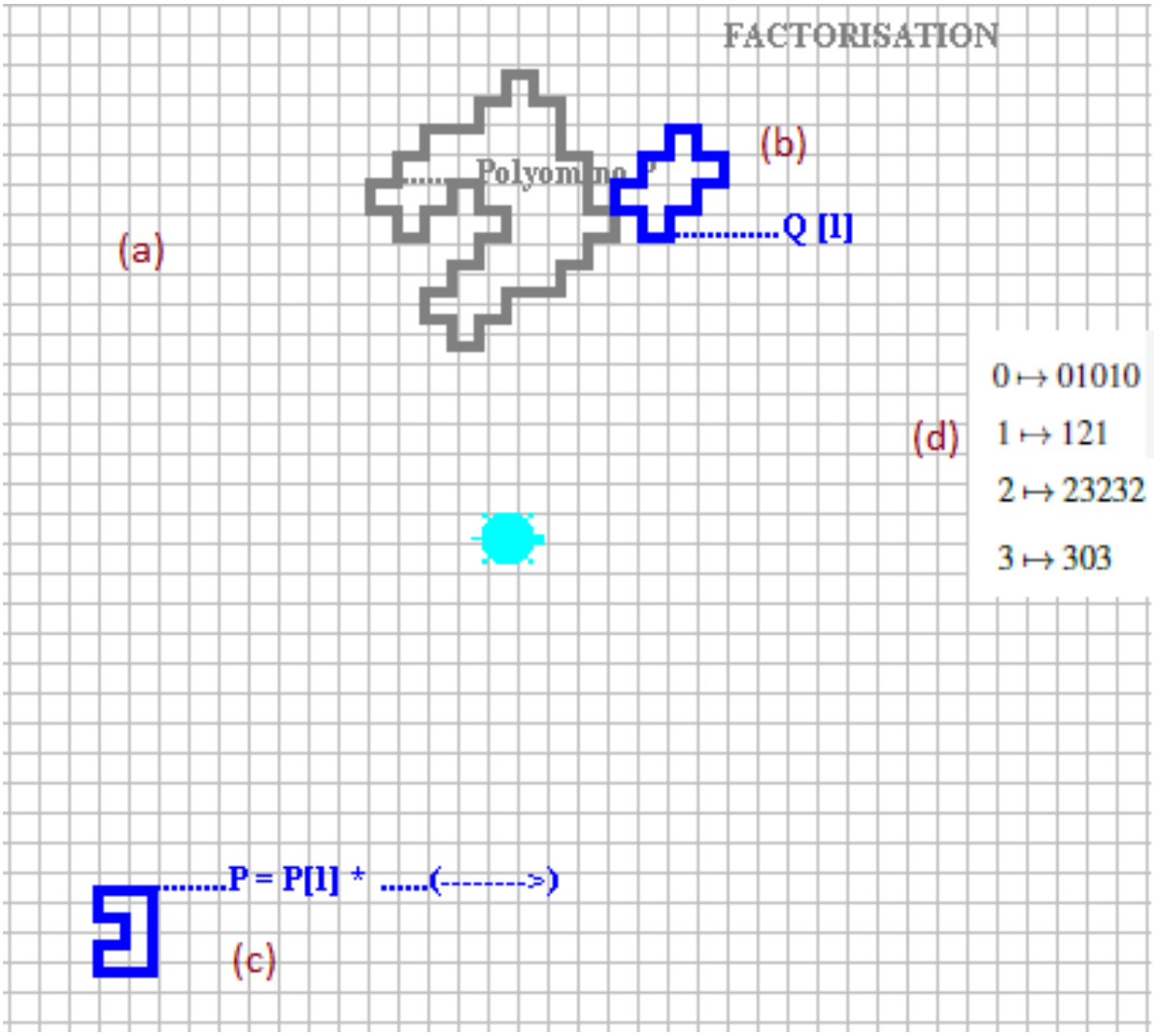


Figure 3.10: Illustration du travail de notre algorithme de factorisation de polyominos (implémentation en Java en utilisant le module Turtle de Python). Soit la factorisation d'une figure discrète composée en forme d'aimant (Figure a) de mot de contour 3030101...32 et de longueur du mot de contour $L = 48$, dans le sens antihoraire. L'algorithme détecte la figure a comme étant l'application du morphisme φ (Figure d), representé par le morphisme parallélogramme Q (Figure c), appliqué au contour de la figure discrète première P en forme de vaisseau spatial (Figure b). Le tout en quelques millisecondes.

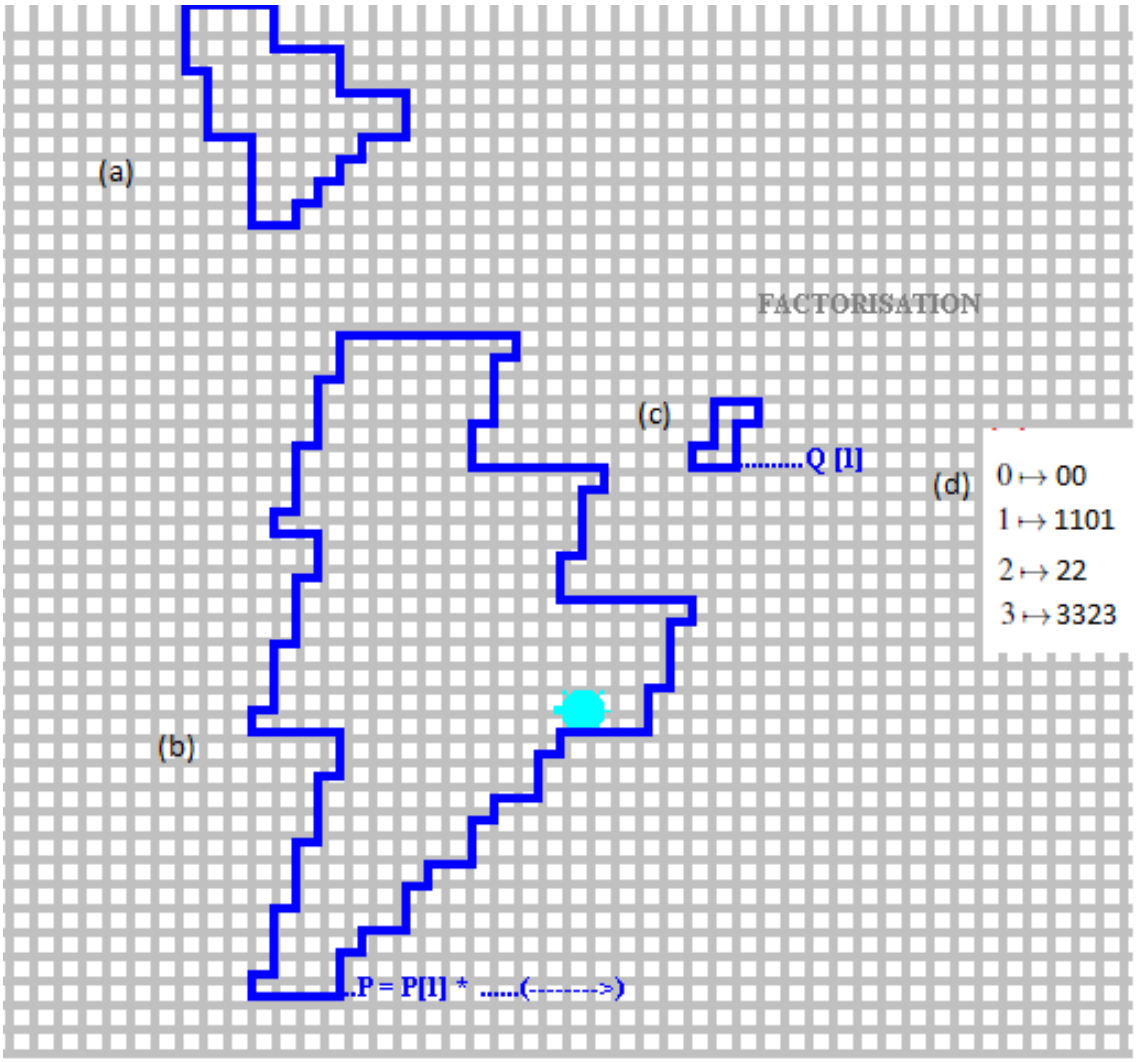


Figure 3.11: Illustration du travail de notre algorithme de factorisation de polyominos (implémentation en Java en utilisant le module Turtle de Python). Soit la factorisation d'une figure discrète composée en forme de sapin (Figure b) de mot de contour 000011...323 et de longueur du mot de contour $L = 120$, dans le sens antihoraire. L'algorithme détecte la figure c comme étant l'application du morphisme φ (Figure d), representé par le morphisme parallélogramme Q (Figure c), appliqué au contour de la figure discrète première P en forme de sapin (Figure a). Le tout en quelques millisecondes.

composé en produit de deux polyominos plus petits.

Théorème 8. *Tout polyomino P peut-être factorisé en produit de polyominos premiers en* $\mathscr{O}(n^5)$ *avec n étant le périmètre de P.*

Démonstration. Soit w le mot de contour de P commençant par $\mathbf{0}$. En choisissant chaque occurrence de $\mathbf{0}$ en w pour construire $\varphi(\mathbf{0})$, il y a au plus n valeurs possibles (et puis $\varphi(\mathbf{2})$ est déterminé). Une fois $\varphi(\mathbf{0})$ est choisi, il y a au plus n valeurs possibles pour $\varphi(\mathbf{1})$ (et puis $\varphi(\mathbf{3})$ est déterminé). Enfin, quand $\varphi(a)$ est connu pour chaque $a \in \mathscr{F}$, il reste à vérifier si w pourrait être décodé de φ, ce qui se fait en temps linéaire tout au plus. Par conséquent, il peut être décidé dans $\mathscr{O}(n^3)$ si w est décodable selon certains morphismes parallélogrammes φ. Le test doit être effectué pour chaque conjugué de w commençant par $\mathbf{0}$, on obtient alors $\mathscr{O}(n^4)$. Enfin, en répétant cette réduction jusqu'à ce que les objets ne soient plus décomposables, on obtient la complexité affirmée $\mathscr{O}(n^5)$. □

3.5 DISCUSSION

La limite $\mathscr{O}(n^5)$ semble assez élevée et il n'est pas évident qu'elle puisse être atteinte en pratique. Pour appuyer cette impression, étudions l'algorithme amelioré pour un polyomino P ayant un mot de contour

$$w = \mathbf{0}^k\mathbf{1}^k\mathbf{2}^{k-1}\mathbf{3}\mathbf{2}\mathbf{3}^{k-1}.$$

Soit $n = |w| = 4k$. Comme P est un carré $(k \times k) - 1$ cellules, on pourrait prouver qu'il est premier. Pour faire en sorte que l'algorithme prenne autant de temps que possible, supposons que k ait d diviseurs. Ensuite l'algorithme va essayer le k conjugué de w commençant par $\mathbf{0}$ et construira d images pour $\varphi(\mathbf{0})$. De même, nous devrons considérer d images possibles pour $\varphi(\mathbf{1})$. Le décodage étant effectué en temps linéaire, nous obtenons une limite globale

Algorithme 3 : Amélioration de l'algorithme naïf

```
1  Données: w is closed , freeman word, simple and self avoidant ;
2  Function FACTORIZE(w :mot de contour)
3  début
4      pour i ∈ OCCURRENCES (0,w) faire
5          /* OCCURRENCES permet de construire les blocs et prend comme argument l ∈ F. Initialement,
              comme argument 0, la fonction construit chaque bloc commençant et se terminant par 0.
              */
6          u ← CONJUGATE (w,i)/* CONJUGATE construit les conjugués φ0 et φ2 pour les chemins horizontaux,
              φ1 et φ3 pour les chemins verticaux.                                              */
7          φ ← FACTORIZERECT (∅,0)/* FACTORIZERECT est imbriquée dans FACTORIZE. La fonction essaie,
              pour φ non complètement défini et φ(0) déterminé, son conjugué φ(2) aussi déterminé,
              chaque bloc possible pour chaque 1 ou 3. Cela permet de décoder le mot de contour pour
              voir φ(w) = w'.                                                                   */
8          si ∅ ≠ φ alors
9              /* Si le morphisme parallélogramme est complètement défini et non trivial, on le
                  retourne. Donc φ(w) = w' qui est un mot de contour fermé et auto-évitant. On a
                  alors une figure discrète composée.                                       */
10             retourner φ
11     retourner ∅ /* Si φ est un ensemble vide alors on peut conclure qu'on a alors une figure discrète
          première.                                                                         */

12 Fonction FACTORIZERECT(w :mot de contour φ : morphism, i : integer)
13 début
14     si i ≥ |w| alors
15         /* Nous vérifions tout d'abord si le décodage est complet.                       */
16         si φ est complètement défini et non trivial. alors
17             retourner φ /* Oui, le décodage est complet et il existe un φ.              */
18         sinon
19             retourner ∅/* Non, le décodage est incomplet. φ est un ensemble vide.       */
20     sinon
21         l ← w[i]
22         si φ(l) est défini alors
23             /* On vérifie si tous les prochains blocs correspondent à φ(l).             */
24             k ← |φ[(l)]|/* On pose quand la longueur du morphisme homologue appliquée sur le chemin
                  vertical ou horizontal.                                                   */
25             si k > |w| − i ou w[i : i+k] alors
26                 /* Le décodage est incomplet ou mauvais si k > |w| − i ou k>w[i : i+k].   */
27                 retourner ∅
28             sinon
29                 /* Autrement, le décodage est bon et on retourne FACTORIZERECT avec comme argument
                      φ et la position i+k.                                                 */
30                 retourner FACTORIZERECT (φ, i+k)
31         sinon
32             /* Nous considérons toutes les constructions possibles du prochain bloc.     */
33             pour j ∈ OCCURRENCES (l, w[i : |w| − 1]) faire
34                 φ(l) ← w[i : j]
35                 φ(l̄) ← ŵ[i : j]
36                 φ ←FACTORIZERECT (φ, j+1)
37                 si φ is not trivial alors
38                     retourner φ
39                 sinon
40                     retourner ∅
41         retourner ∅
```

$\mathcal{O}(kd^2n) = \mathcal{O}(n^2d^2)$. Cependant d est en général beaucoup plus petit que k. Dès lors, il est facile de voir par exemple que $d \leq \sqrt{k}$ (limites strictes de la théorie des nombres). Par conséquent, on obtient dans ce cas une limite de $\mathcal{O}(n^3)$.

Nous pensons qu'une amélioration significative pourrait réduire la limite théorique à $\mathcal{O}(n^4)$ ou même $\mathcal{O}(n^3)$ en prenant en compte les répétitions de certains facteurs. Par exemple, quand nous essayons de factoriser le polyomino P du chapitre 4 par $\varphi(\mathbf{0}) = \mathbf{0}$ et que nous lisons le facteur $\mathbf{0}^k$, il serait plus efficace de garder en mémoire le fait que $\varphi(\mathbf{0})$ ne peut etre construit que par des puissances de $\mathbf{0}$ qui divisent k.

CHAPITRE 4

EXPLORATION INFORMATIQUE

Dans ce chapitre, nous nous intéressons à l'évaluation pratique des deux algorithmes de factorisation décrits au chapitre précédent. Nous expliquons de quelle façon un banc d'essai est généré, puis nous proposons une analyse comparative de nos deux algorithmes de factorisation des polyominos.

4.1 MÉTHOLOGIE

4.1.1 JAVA

Pour l'évaluation expérimentale, notre choix s'est porté sur le langage de programmation **Java**. Afin de visualiser les objets construits, nous avons choisi le langage de programmation **Python** avec son module graphique « **Turtle** », très adapté pour illustrer les figures discrètes. L'interface entre les deux langages s'est faite par l'intermédiaire du langage Jython, qui est un interpréteur Python écrit en Java.

Pour chronométrer les différents temps de calcul, nous utilisons la bibliothèque **java.lang. management**, qui offre la fonction $currenttime()$ (voir algorithme 4.1), c'est-à-dire l'heure du

système actuel qui est déterminée immédiatement avant et après l'exécution de l'algorithme. On atténue ainsi l'impact des tâches de traitement de fond sur le temps de fonctionnement. La valeur retournée en minutes, secondes, millisecondes ou nanosecondes représente le temps fixe d'exécution.

Typiquement, pour une étude comparative de la complexité en temps, la plupart des programmes sont codés similairement en introduisant des fragments de code ressemblant à ce qui se trouve à l'algorithme 4.1.

```
long t1 = System.currentTimeMillis();
task.run();    // task is a Runnable which encapsulates the unit of work
long t2 = System.currentTimeMillis();
System.out.println("My task took " + (t2 - t1) + " milliseconds to execute.");
```

Figure 4.1: Algorithme "currentTimeMillis()"

4.1.2 MÉTRIQUES D'ÉVALUATION

La notation *grand-O* décrit le comportement d'une fonction lorsque l'argument tend vers l'infini, généralement à l'aide de fonctions simples (voir par exemple la référence classique [Cormen et al., 2001] pour plus de détails).

Lorsqu'on évalue les performances d'un algorithme de façon empirique, on doit utiliser différentes mesures pour estimer le comportement asymptotique du temps d'exécution. On suppose dans ce chapitre que tous les temps de calcul suivent une règle de puissance de la forme $p \sim kn^a$.

Il est alors possible d'estimer la valeur du coefficient a en prenant deux temps d'exécution $t1$ et $t2$ ainsi que les deux tailles des instances $n1$ et $n2$, qui doivent vérifier la relation

$$\frac{t_2}{t_1} = \left(\frac{n_2}{n_1}\right)^a.$$

On peut alors facilement estimer a à l'aide de la formule

$$a = \log\left(\frac{t2}{t1}\right) / \log\left(\frac{n2}{n1}\right).$$

Voir [Goldreich, 2008] pour plus de détails sur l'estimation empirique de la complexité.

Nous nous intéressons donc à estimer *l'ordre de complexité local* par rapport à la notation grand-O. Nous nous attendons donc à obtenir une borne près de $\mathscr{O}(n^7)$ pour l'algorithme naïf et $\mathscr{O}(n^5)$ pour l'algorithme amélioré. Si l'ordre de complexité local suit une règle de puissance quelconque alors la valeur empirique doit rester constante en fonction des données en entrée.

Considérant la puissance de calcul de l'ordinateur et la méthode d'implémentation des algorithmes (style du codage, techniques utilisées par le développeur, langages de programmation), la borne empirique peut être très différente de la borne théorique. C'est pourquoi, il est plus important de retenir que l'évaluation empirique de a servira toujours d'indicateur de comparaison locale de la performance de nos deux algorithmes de factorisation même si une borne théorique n'est pas atteinte [Greene et Knuth, 2007].

Pour l'évaluation empirique, on prend comme paramètres le temps d'exécution et l'ordre de complexité local de nos deux algorithmes de factorisation. Nous testons donc sur un très grand nombre de jeux de données nos deux algorithmes de factorisation en fonction de la longueur du mot de contour.

4.2 BANC D'ESSAI

À l'instar des graphes (orientés et non orientés), l'énumération de polyominos est un problème très difficile et il semble très peu probable qu'il existe un algorithme efficace permettant de tirer de façon complètement aléatoire un polyomino de périmètre p, pour une valeur de p

donnée.

Nous devons donc nous rabattre sur une stratégie de génération qui ne garantit par l'uniformité. Nous avons choisi de générer 250 polyominos premiers de longueur de mot de contour allant de 10 à 200 et 1000 polyominos composés de longueur de mot de contour allant de 100 à 900. Les polyominos premiers sont générés pseudo-aléatoirement. Les polyominos composés sont générés sur la base de polyominos premiers, auxquels on applique un morphisme parallélogramme φ lui aussi généré de façon aléatoire.

À cet effet, nous développons deux générateurs aléatoires :

1. un générateur pseudo-aléatoire de polyominos, que nous dénotons par GPoly ;
2. un générateur pseudo-aléatoire de polyominos parallélogrammes, que nous dénotons par GPara.

4.2.1 GÉNÉRATION DE POLYOMINOS

Il s'avère nécessaire de concevoir un générateur de polyominos premiers. En effet, on ne peut énumérer les polyominos de façon manuelle ou déterministe puisqu'il y en a un nombre dont la croissance est exponentielle. Notre banc d'essai est donc pseudo-aléatoire à défaut d'être complètement aléatoire.

Il y a différentes façons de résoudre le problème de la construction du générateur. Il convient de commencer par la solution la plus intuitive. L'idée de base de cette méthode simple est la suivante : étant donné un polyomino P, soit un générateur aléatoire G qui prend en entrée un nombre entier n qui correspond au nombre de cellules à générer des polyominos.

Comme le montre la figure 4.2 ci-dessus, on construit un polyomino de n cellules en lui ajoutant au hasard une cellule sur son contour. Cette opération est effectuée jusqu'à ce qu'on

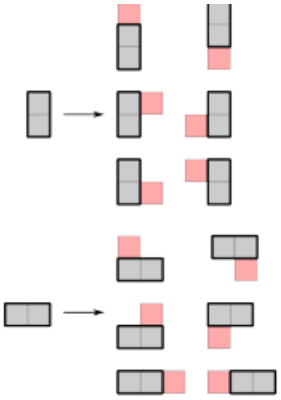


Figure 4.2: Ajout d'une cellule aléatoire à un polyomino.

ait placé le bon nombre de cellules. L'ensemble de cellules ainsi généré est clairement connexe.

Clairement, un polyomino ainsi généré peut être premier ou composé, mais, dans la plupart des cas, il s'agit d'un polyomino premier. Pour cette raison, nous proposons un deuxième générateur dans la sous-section suivante, qui nous permet de construire un échantillon de polyominos composés.

4.2.2 GÉNÉRATION DE POLYOMINOS COMPOSÉS

Rappelons qu'un polyomino non unitaire P est appelé *composé* s'il existe un mot de contour quelconque u et un morphisme parallélogramme quelconque φ tel que $P = \varphi(u)$. En d'autres termes, s'il peut être décomposé comme le produit d'un morphisme parallélogramme et d'un autre polyomino. Nous nous chargeons de générer pseudo-aléatoirement le polyomino parallélogramme φ puisque le polyomino composé est construit par $P = \textsc{Poly}(\varphi)$ et que l'algorithme de génération de polyominos premiers se charge de générer le mot de contour quelconque u de $P = \textsc{Poly}(\varphi)$.

Notre algorithme (voir algorithme 5) commence en générant les morphismes homologues qui

Algorithme 4 : Générateur pseudo-aléatoire de polyominos

```
1  début
2      NbrePolyominos = ALEATOIRE(40) /* Recursivement générer un nombre x de
           polyominos possédant chacun un nombre y de cellules                    */
3      cellules = ALEATOIRE(Gamme saisie utilisateur)
4      /* Entrez la portée du nombre de cellules à générer                        */
5      pour Polyominos_Gnrs ← 1 à |NbrePolyominos| faire
6          pour i ← 1 à |cellules| faire
7              chemin = ALEATOIRE(←,→,↑,↓)
8              si chemin ==→ alors
9                  Contour (→).carrunitaire (↑,←,↓,→)
10             sinon si chemin ==↑ alors
11                 Contour (↑).carrunitaire (↑,←,↓,→)
12             sinon si chemin ==← alors
13                 Contour (←).carrunitaire (↑,←,↓,→)
14             sinon si chemin ==↓ alors
15                 Contour (↓).carrunitaire (↑,←,↓,→)
16 Fonction Contour(int chemin )) /* Extension aléatoire des arêtes :←,→,↑,↓      */
17 début
18     x_cell = 0
19     y_cell = 0
20     si chemin ==→ alors
21         x_cell += 10
22     sinon si chemin ==↑ alors
23         y_cell += 10
24     sinon si chemin ==← alors
25         x_cell −= 10
26     sinon si chemin ==↓ alors
27         y_cell −= 10
28     SetPos(x_cell, y_cell)
29 Fonction carrunitaire((↑,←,↓,→ )) /* Construire un carré unitaire pour chaque
       extension                                                                  */
30 début
31     int[]carr unitaire = [↑,←,↓,→] pour i ← 1 à |4| faire
32         si carr unitaire[i] ==→ alors
33             heading(90) forward(10)
34         sinon si carr unitaire[i] ==↑ alors
35             heading(360) forward(10)
36         sinon si carr unitaire[i] ==← alors
37             heading(270) forward(10)
38         sinon si carr unitaire[i] ==↓ alors
39             heading(180) forward(10)
```

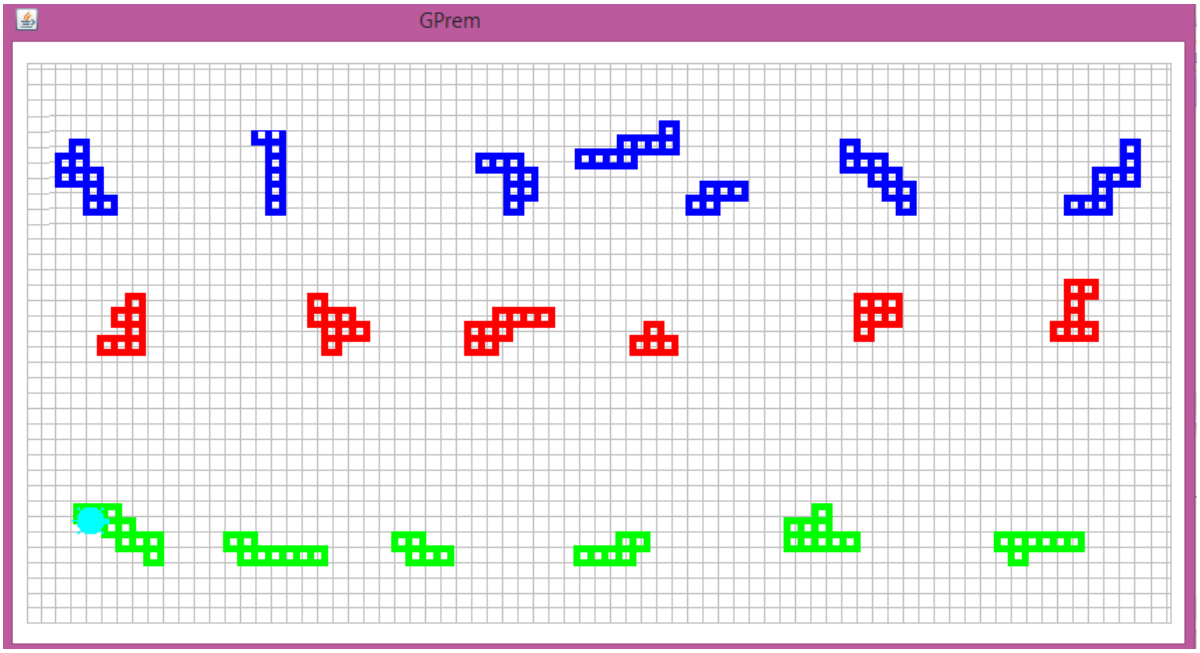


Figure 4.3: GPoly. Banc d'essai disponible dans l'archive électronique jointe comme matériel complémentaire.

remplacent par leur chemin arbitraire le pas élémentaire horizontal $\varphi(\mathbf{0})$ et vertical $\varphi(\mathbf{1})$. La longueur des chemins $\varphi(0)$ et $\varphi(1)$ est spécifiée selon des paramètres. Nous nous assurons que le morphisme est homologue et que le polyomino est parallélogramme. Nous nous assurons également que les chemins arbitraires verticaux et horizontaux sont auto-évitants et fermés.

Plus précisément, nous suivons les étapes ci-bas :

1. Soit n le nombre de polyominos parallélogrammes à générer ;
2. Soit la longueur aléatoirement générée ℓ ;
3. Pour $i \in [1,n]$, nous construisons respectivement les chemins y_i et z_i tel que $y_i = \varphi_i(\mathbf{0})$ et $z_i = \varphi_i(\mathbf{1})$;
4. Pour $i \in [1,n]$, nous construisons $w'_i = y_i \cdot z_i$;
5. Nous créons l'image du chemin inverse $\varphi(\overline{w'_i})$;
6. Nous vérifions que le résultat est auto-évitant.

Algorithme 5 : Générateur pseudo aléatoire de polyominos Composés

```
début
    Polyomino para = 1
    pour Polyomino para ← 1 à |ALEATOIRE(Gamme saisie utilisateur)| faire
        MorphismeGen ()
Fonction MorphismeGen()
début
    l_0 = ALEATOIRE(Gamme saisie utilisateur)
    l_1 = ALEATOIRE(Gamme saisie utilisateur)
    w_0 = |l_0|
    w_1 = |l_1|
    w_0[0] = [0]
    w_1[0] = [1]
    w_0[l_0] = [0]
    w_1[l_1] = [1]
    /* || signifie ''ou''                                                    */
    pour i ← 1 à |w_0|||w_1| faire
        PAS ALEATOIRE_{0||1} = ALEATOIRE(←,→,↑,↓)
        si i > 1 alors
            w_(0,1)[i] = PAS ALEATOIRE_(0,1)
        si w_(0||1)[i] == 0 & w_(0||1)[i−1] == 2 alors
            i = i − 1
        si w_(0||1)[i] == 2 & w_(0||1)[i−1] == 0 alors
            i = i − 1
        si w_(0||1)[i] == 1 & w_(0||1)[i−1] == 3 alors
            i = i − 1
        si w_(0||1)[i] == 3 & w_(0||1)[i−1] == 1 alors
            i = i − 1
        si i == w_{0||1}[|w_{0||1}| − 2] & w_{0||1}[|w_{0||1}| − 2] == 2 alors
            i = i − 1
    w_2 = ŵ_0
    w_3 = ŵ_1
    φ = w_0 · w_1 · w_2 · w_3
    Booleanfermé + auto − évitant = fermé + auto − évitant(φ)
    si fermé + auto − évitant == vrai alors
        Afficherφ
```

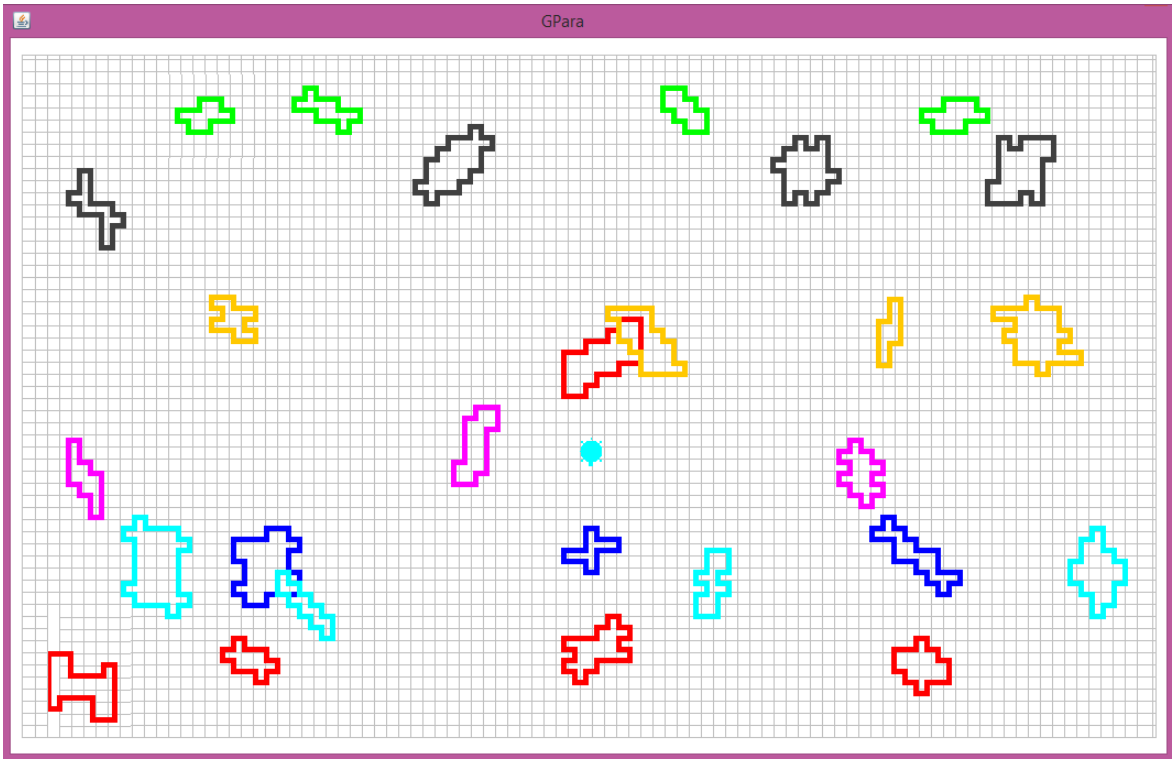


Figure 4.4: GPara. Banc d'essai disponible dans l'archive électronique jointe comme matériel complémentaire.

4.3 RÉSULTATS EMPIRIQUES

À la figure 4.5, on peut y lire le temps d'exécution des algorithmes de factorisation sur des polyominos de longueur de mot de contour ($\ell < 50$) :

1. Pour l'algorithme naïf, on note une fonction à croissance lente et soutenue.
2. Pour l'algorithme amélioré, on note une fonction croissante avec une phase de croissance rapide, $28 < \ell < 40$ et une phase de plateau, $\ell < 25$.

Au tableau 4.1, on décrit l'ordre de complexité empirique local des polyominos de longueur de mot de contour ($\ell < 50$) :

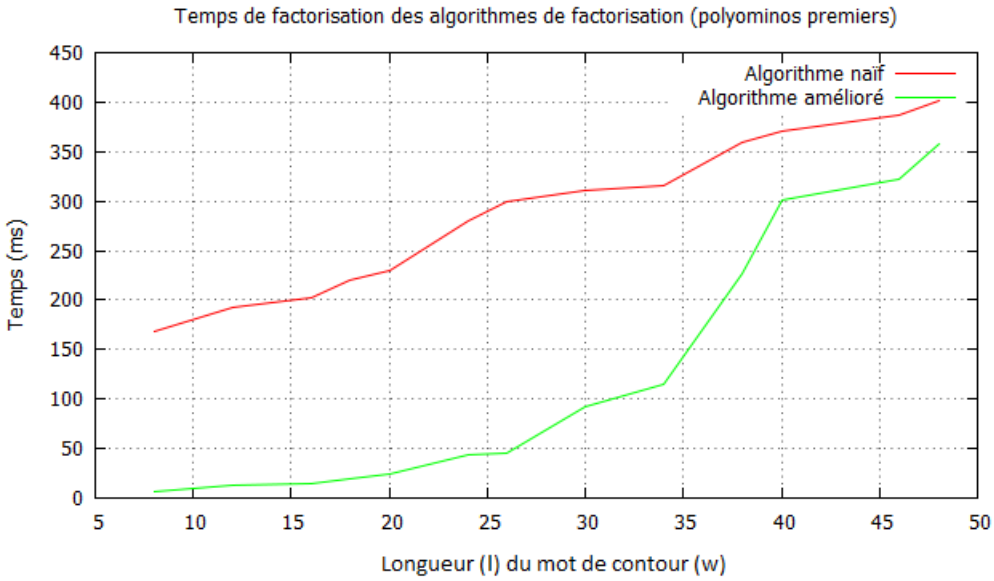


Figure 4.5: Temps de factorisation (polyominos premiers)

l	Algorithme naïf (en ms)	Ordre de complexité local (l^a)	Algorithme amélioré (en ms)	Ordre de complexité local (l^a)
20	230	2.7	25	0.6
30	313	1.54	93	0.36
40	375	1.41	300	0.27
45	386	1.83	321	0.29

Tableau 4.1: Ordre de complexité empirique local (polyominos premiers)

1. On voit clairement que l'algorithme naïf présente un ordre linéaire de la croissance ne suivant pas une règle de puissance.
2. On voit clairement que l'algorithme amélioré présente un ordre linéaire de la croissance ne suivant pas une règle de puissance.
3. Localement, l'ordre de complexité de l'algorithme amélioré est plus faible que celui de l'algorithme naïf.

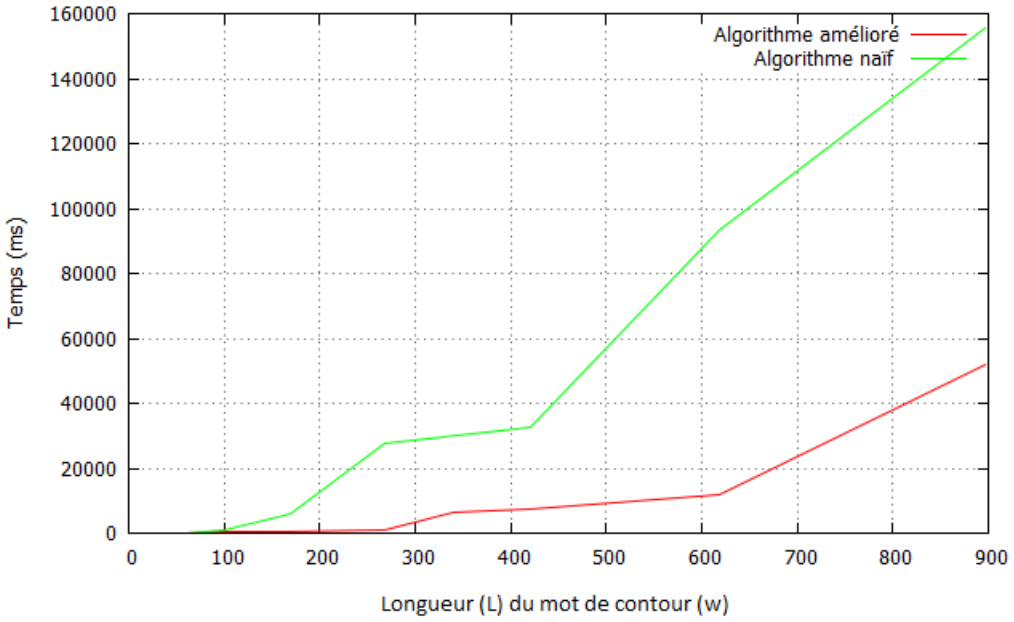


Figure 4.6: Temps de factorisation (polyominos composés)

À la figure 4.6), on peut lire les temps d'éxécution de tests de primalité pour des polyominos composés de longueur $\ell < 900$:

1. Pour l'algorithme naïf, on observe des croissances rapides, $0 < \ell < 100$, $260 < \ell < 420$, et des phases de plateau, $180 < \ell < 270$, $430 < \ell < 900$.
2. Pour l'algorithme amélioré, on note une fonction croissante avec des phases de croissance rapide, $170 < \ell < 260$, $420 < \ell < 900$, et des phases de plateau, $10 < \ell < 280$,

$260 < \ell < 620$.

L	Algorithme naïf (en *ms*)	Ordre de complexité local (L^a)	Algorithme amélioré (en *ms*)	Ordre de complexité local (L^a)
20	104	0.3	13	0.15
40	156	1.7	16	0.29
60	218	1.21	16	0.01
100	1060	0.32	412	0.15
170	6098	0.3	494	2.92
268	27653	0.3	1143	0.54
420	32389	2.86	6320	0.26
618	73530	0.47	11754	0.62
896	155538	0.49	51630	0.25

Tableau 4.2: Ordre de complexité empirique local (polyominos composés)

Au tableau 4.2, on trouve l'ordre de complexité empirique local des polyominos composés de longueur de mot de contour $\ell < 900$:

1. On voit clairement que l'algorithme naïf présente un ordre linéaire de la croissance ne suivant pas une règle de puissance.
2. On voit clairement que l'algorithme amélioré présente un ordre linéaire de la croissance ne suivant pas une règle de puissance.
3. Localement, l'ordre de complexité de l'algorithme amélioré est plus faible que celui de l'algorithme naïf.

4.4 ANALYSE DES RÉSULTATS EMPIRIQUES

On note les caractéristiques et conclusions suivantes sur le comportement local des algorithmes de factorisation pour des mots de contour :

1. Pour des polyominos premiers et composés, dans les deux cas, on a des fonctions linéaires globalement croissantes en fonction de la longueur du mot de contour.

2. Pour des polyominos premiers ou composés, l'algorithme amélioré est beaucoup plus efficace en temps de factorisation que l'algorithme naïf.
3. L'indicateur a dénote d'une meilleure performance de l'algorithme de factorisation amélioré par rapport à l'algorithme de factorisation naïf.
4. Pour des polyominos premiers ou composés, l'algorithme naïf et amélioré ne suivent pas de règle de puissance empirique locale. On en déduit que la la borne empirique n'est pas atteinte puisque a varie selon les données d'entrées. On en conclut que les données générées sont insuffisantes pour atteindre la borne empirique.

4.5 CONCLUSION

En clair, nous savons que la borne empirique pour nos deux algorithmes de factorisation est loin d'être atteinte. Il nous faut beaucoup plus de données en entrée. Il est probable également que les algorithmes de génération génèrent trop rarement les cas les plus défavorables. Au niveau empirique local, on n'a pas pu atteindre la borne empirique mais clairement, la comparaison empirique locale nous montre que l'algorithme amélioré est plus performant en temps d'exécution, ce qui est cohérent avec les bornes théoriques démontrées.

L'explication de la performance empirique locale de l'algorithme amélioré est la suivante :

1. En choisissant chaque occurrence de $\mathbf{0}$ en w pour construire $\varphi(\mathbf{0})$, soit k_0 le premier conjugé de w commençant par $\mathbf{0}$ et construisant d images pour $\varphi(\mathbf{0})$.
2. Soit k_0 determiné en choisissant chaque occurrence de $\mathbf{1}$ en w pour construire $\varphi(\mathbf{1})$, soit k_1 le premier conjugé de w commençant par $\mathbf{1}$ et construisant d images pour $\varphi(\mathbf{1})$.
3. Soit N le nombre de tentatives globales pour construire les k conjugués et leur images de w commençant par $\mathbf{0}$ et $\mathbf{1}$.

La fonction $skip()$, avec comme séquence d'entrée le mot de contour w, retourne une séquence qui ignore N lettres de la séquence sous-jacente. Pour que l'algorithme soit le plus rapide possible, supposons que les quatre images de φ sont choisies, la probabilité $prob(k_{0,1})$ que le premier essai factorise P et que u et v surviennent de façon contiguë est très forte puisque $N \leq skip(|w|, k_{0,1})$ pour un certain $\ell < 60$. Dès lors, on obtient un temps global constant de factorisation $\mathbf{t}$ des polyominos composés pour des longueurs de contour $\ell < 60$.

En conclusion, l'évaluation expérimentale nous a permis de mieux comprendre certaines caractéristiques spécifiques du fonctionnement des algorithmes de factorisation des polyominos au niveau empirique que l'analyse théorique ne peut déceler.

CONCLUSION

L'objet de ce mémoire concernait les polyominos qui pavent le plan par translation et, en particulier le problème de leur factorisation. En présentant une solution sur la primalité des figures discrètes, nous avons mis en lumière l'intérêt de nos recherches et sa potentielle application dans le domaine de la cryptographie à clé publique/privée. En effet, de futures recherches dans cette direction permettront l'application de la primalité des figures discrètes par leurs mots de contour à la cryptographie asymétrique. Cependant, il faudra se pencher sur l'application de notions mathématiques propres à l'arithmétique élémentaire.

Dans ce mémoire, pour la génération de notre banc d'essai, nous avons également implémenté deux générateurs pseudo-aléatoires de polyominos et de polyominos composés. Nous avons aussi fourni deux algorithmes permettant d'exprimer, en temps polynomial, tout polyomino comme le produit de polyominos premiers. En conséquence, il s'ensuit que nous pouvons décider si un polyomino est premier ou composé en temps polynomial.

L'algorithme amélioré est actuellement le meilleur algorithme connu pour résoudre le problème de la factorisation de figures discrètes. On note aussi comme mentionné ci-dessus que la limite $\mathcal{O}(n^5)$ est plutôt grande et il ne serait pas surprenant de concevoir dans un futur proche des algorithmes plus efficaces pour résoudre le problème de la factorisation de façon plus efficace.

Par exemple, on pourrait essayer de factoriser le mot de contour en trouvant de manière récursive les plus longues paires de facteurs homologues et en cassant ainsi le mot de contour du polyomino en plus petits morceaux jusqu'à ce qu'une factorisation soit trouvée.

Il reste de nombreux problèmes à résoudre dans les domaines étudiés comme la notion de congruence et de division euclidienne de polyominos mais nous sommes convaincus que les outils présents dans ce mémoire favoriseront la résolution de certains d'entre eux.

Les travaux presentés dans ce mémoire ont fait l'objet d'un article à la conférence LATA 2014.

Le lecteur soucieux de découvrir et d'exécuter nos algorithmes de factorisation, nos générateurs de polyominos et le banc d'essai, est invité à regarder les codes source à la disposition du public hébergé sur Bitbucket [1], qui ne nécessite qu'une installation de la plateforme Java SE (Edition Standard).

1. `https://bitbucket.org/atall/polyosarithmetic/downloads`

BIBLIOGRAPHIE

[Beauquier et Nivat, 1991] Beauquier, D. et M. Nivat. 1991. « On translating one polyomino to tile the plane », *Discrete Comput. Geom.*, vol. 6, p. 575–592.

[Berthé et Vuillon, 2000] Berthé, V. et L. Vuillon. 2000. « Tilings and rotations on the torus : a two-dimensional generalization of Sturmian sequences », *Discrete Mathematics*, vol. 223, no. 1, p. 27–53.

[Blondin Massé et al., 2009] Blondin Massé, A., S. Brlek, A. Garon, et S. Labbé. 67–78, 2009. « Christoffel and Fibonacci tiles ». In Brlek, S., X. Provençal, et C. Reutenauer, éditeurs, *DGCI 2009, 15th IAPR Int. Conf. on Discrete Geometry for Computer Imagery*. Coll. « Springer-Verlag LNCS 5810 ».

[Blondin Massé et al., 2012] Blondin Massé, A., S. Brlek, et S. Labbé. 2012. « A parallelogram tile fills the plane by translation in at most two distinct ways », *Discrete Applied Mathematics*, vol. 160, no. 7, p. 1011–1018.

[Blondin Massé et al., 2013] Blondin Massé, A., A. Garon, et S. Labbé. 2013. « Combinatorial properties of double square tiles », *Theoretical Computer Science*, vol. 502, p. 98–117.

[Blondin Massé et al., 2014] Blondin Massé, A., A. M. Tall, et H. Tremblay. 2014. *On the Arithmetics of Discrete Figures*. Coll. « Language and Automata Theory and Applications »,

p. 198–209. Springer.

[Brass et al., 2005] Brass, P., W. O. Moser, et J. Pach. 2005. *Research problems in discrete geometry*. Springer.

[Brlek et al., 2009a] Brlek, S., M. Koskas, et X. Provençal. 2009a. « A linear time and space algorithm for detecting path intersection ». In *Discrete Geometry for Computer Imagery*, p. 397–408. Springer.

[Brlek et al., 2005] Brlek, S., G. Labelle, et A. Lacasse. 2005. « A note on a result of Daurat and Nivat ». In *Developments in Language Theory*, p. 189–198. Springer.

[Brlek et Provençal, 2006] Brlek, S. et X. Provençal. 2006. « On the problem of deciding if a polyomino tiles the plane by translation ». In Holub, J. et J. Žďárek, éditeurs, *Proceedings of the Prague Stringology Conference '06*. Coll. « ISBN80-01-03533-6 », p. 65–76, Prague, Czech Republic. Czech Technical University in Prague.

[Brlek et al., 2009b] Brlek, S., X. Provençal, et J.-M. Fédou. 2009b. « On the tiling by translation problem », *Discrete Applied Mathematics*, vol. 157, no. 3, p. 464–475.

[CAMINOS, 2014] CAMINOS. 2014. Disponible à `http://www.trictrac.net/actus/caminos-faites-votre-chemin-`.

[Cormen et al., 2001] Cormen, T. H., C. E. Leiserson, R. L. Rivest, C. Stein, et al. 2001. *Introduction to algorithms*. T. 2. MIT press Cambridge.

[Dagorne, 1989] Dagorne, D. 1989. « Modélisation géométrique en télédétection : superposition et rectification de données satellitaires et spatialisées ».

[DGCI, 2014] DGCI. 2014. « The 18th international conference on discrete geometry for computer imagery ». Disponible à `http://dgci-conference.org/`.

[Diffie et Hellman, 1976] Diffie, W. et M. Hellman. 1976. « New directions in cryptography », *IEEE Transactions on Information Theory*, vol. 22, no. 6, p. 644–654.

[Dikau, 1989] Dikau, R. 1989. « The application of a digital relief model to landform analysis in geomorphology », *Three dimensional applications in geographical information systems*, p. 51–77.

[Franke et al., 2005] Franke, J., T. Kleinjung, C. Paar, J. Pelzl, C. Priplata, et C. Stahlke. 2005. *SHARK : A realizable special hardware sieving device for factoring 1024-bit integers*. Coll. « Cryptographic Hardware and Embedded Systems–CHES 2005 », p. 119–130. Springer.

[Gambini et al., 2007] Gambini, I., L. Vuillon, et al. 2007. « An algorithm for deciding if a polyomino tiles the plane by translations », *Theoretical Informatics and Applications*, vol. 41, p. 147–155.

[Glassner, 1995] Glassner, A. S. 1995. *Principles of digital image synthesis : Vol. 1*. T. 1. Elsevier.

[Goldreich, 2008] Goldreich, O. 2008. « Computational complexity : a conceptual perspective », *ACM SIGACT News*, vol. 39, no. 3, p. 35–39.

[Golomb, 1996] Golomb, S. 1996. *Polyominoes, second edition*. New Jersey : Princeton University Press.

[Golomb, 1970] Golomb, S. W. 1970. « Tiling with sets of polyominoes », *Journal of Combinatorial Theory*, vol. 9, no. 1, p. 60 – 71.

[Greene et Knuth, 2007] Greene, D. H. et D. E. Knuth. 2007. *Mathematics for the Analysis of Algorithms*. Springer.

[Gusfield, 1997] Gusfield, D. 1997. *Algorithms on strings, trees and sequences : computer science and computational biology*. Cambridge University Press.

[Gusfield et Stoye, 2004] Gusfield, D. et J. Stoye. 2004. « Linear time algorithms for finding and representing all the tandem repeats in a string », *Journal of Computer and System Sciences*, vol. 69, no. 4, p. 525–546.

[IARP, 2014] IARP. 2014. « International association of pattern recognition », *Discrete geometry for computer imagery 16th IAPR International Conference, DGCI, Nancy, France, April 6-8, 2011 : proceedings. Berlin : Springer, 2011.* `http://www.iapr.org/`.

[KALEIDOSCOPE, 2014] KALEIDOSCOPE. 2014. Disponible à `http://www.jeuxadeux.com/jeux/kaleidoscope.php`.

[Knuth, 2000] Knuth, D. E. 2000. Dancing links. `http://arxiv.org/abs/cs/0011047`.

[Labbé, 2011] Labbé, S. 2011. Tiling solver in Sage. `http://www.sagemath.org/doc/reference/combinat/sage/combinat/tiling.html`.

[Lacassagne et al., 1998] Lacassagne, L., F. Lohier, P. Garda, U. Pierre, et M. Bâtiment. 1998. « Real time execution of optimal edge detectors on risc and dsp processors », *BENCHMARKING*, vol. 2, p. 2.

[Lenstra et al., 1993] Lenstra, A. K., H. W. Lenstra Jr, M. S. Manasse, et J. M. Pollard. 1993. *The number field sieve*. Springer.

[Lothaire, 1983] Lothaire, M. 1983. *Combinatorics on words*. Cambridge University Press.

[Lu et al., 2003] Lu, J., K. N. Plataniotis, et A. N. Venetsanopoulos. 2003. « Face recognition using kernel direct discriminant analysis algorithms », *Neural Networks, IEEE Transactions on*, vol. 14, no. 1, p. 117–126.

[Miller, 1976] Miller, G. L. 1976. « Riemann's hypothesis and tests for primality », *Journal of computer and system sciences*, vol. 13, no. 3, p. 300–317.

[Moore et Robson, 2001] Moore, C. et J. M. Robson. 2001. « Hard tiling problems with simple tiles », *Discrete & Computational Geometry*, vol. 26, no. 4, p. 573–590.

[Morrison, 1968] Morrison, D. R. 1968. « Patricia—practical algorithm to retrieve information coded in alphanumeric », *Journal of the ACM (JACM)*, vol. 15, no. 4, p. 514–534.

[Polyá, 1969] Polyá, G. 1969. « On the number of certain lattice polygons », *Journal of Combinatorial Theory*, vol. 6, no. 1, p. 102 – 105.

[PRIB, 2014] PRIB. 2014. « The 9th iapr conference on pattern recognition in bioinformatics ». Disponible à `http://prib2014.scilifelab.se/`.

[Provençal, 2008] Provençal, X. 2008. « Combinatoire des mots, géométrie discrète et pavages ». Thèse de Doctorat, D1715, Université du Québec à Montréal.

[Richardson, 2007] Richardson, L. F. 2007. *Weather prediction by numerical process*. Cambridge University Press.

[Rivest et al., 1978] Rivest, R. L., A. Shamir, et L. Adleman. 1978. « A method for obtaining digital signatures and public-key cryptosystems », *Communications of the ACM*, vol. 21, no. 2, p. 120–126.